O'REILLY®

TURING 图灵程序设计丛书



Web安全开发指南

掌握白帽子的Web安全技能，从源头消除安全隐患，打造安全无虞的Web应用

Security for Web Developers

[美] John Paul Mueller 著
温正东 译

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

译者介绍

温正东

华南理工大学计算机硕士，曾任华为公司IT工程师，现为富途证券Web运营和基础架构组负责人。

TURING 图灵程序设计丛书

Web 安全开发指南

Security for Web Developers

[美] John Paul Mueller 著
温正东 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北 京

图书在版编目（C I P）数据

Web安全开发指南 / (美) 约翰·保罗·米勒
(John Paul Mueller) 著 ; 温正东译. -- 北京 : 人民
邮电出版社, 2017.6
（图灵程序设计丛书）
ISBN 978-7-115-45408-9

Ⅰ. ①W… Ⅱ. ①约… ②温… Ⅲ. ①互联网络－安全
技术－指南 Ⅳ. ①TP393.408-62

中国版本图书馆CIP数据核字(2017)第083702号

内 容 提 要

本书分为 5 大部分，共 17 章，详细介绍了 Web 安全开发的必备知识，旨在让前端开发人员、设计师、产品经理等前端开发相关人士了解新形势下的安全技能，涉及从最新的智能手机到老旧的台式计算机等各种设备，并且不限定平台。具体内容包括：制订安全计划，运用成功的编码实践，创建有用及高效的测试策略，实现维护周期，查找安全资源。

本书适合所有 Web 开发领域的专业人士阅读。

◆ 著　　　　[美] John Paul Mueller
译　　　　温正东
责任编辑　岳新欣
执行编辑　赵瑞琳
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：17
字数：402千字　　　　2017年 6 月第 1 版
印数：1 – 3 500册　　　2017年 6 月北京第 1 次印刷
著作权合同登记号　图字：01-2017-7589号

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第 8052 号

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

献给那些帮我恢复身体健康的医学专家——他们聆听了我的疾苦并找到了解决方法。是的，我确实需要听从他们的建议。身体健康是一件非常棒的礼物。

目录

第二部分　运用成功的编码实践

第三部分　创建有用及高效的测试策略

第四部分 实现维护周期

第五部分 查找安全资源

前言

勒索软件、病毒、分布式拒绝服务（distributed denial-of-service，DDoS）攻击、中间人攻击、安全漏洞，这些词都会勾起参与应用程序管理的人噩梦般的回忆。现在已经到了这样的地步：在处理关乎应用程序或其相关数据安全的问题时，任何人都会变得风声鹤唳、极其保守。你肯定不希望承担应用程序安全相关的责任，但这是在所难免的。

任何一种安全失误所造成的灾难性后果都会困扰你的余生，让你承受极大的压力。与大多数错误不同的是，你不能将这种错误隐藏起来，因为它会出现在所有人都能看到的行业媒体上。虽然你的名字不会成为安全事故的代名词，但安全问题仍会给你带来很多麻烦，比如官司缠身、失业等。那么应该如何处理这个问题呢？

逃避并不能解决问题，至少不是长期的解决方案。本书并不打算介绍我们可能会遇到的每一种安全威胁或解决它们的方法，而是提供独立解决任何一种安全问题所需要的指导原则和工具，让你看到成功的希望。本书的真正目的是教你如何将事情做对，从而可以安心地睡个好觉。

本书预览

本书会提供处理应用程序安全问题所需的资源。是的，你还会在书中看到一些关于平台的信息，因为浏览器是在特定的平台上运行的。此外，你可能会看到使用桌面端应用程序时出现的一些安全问题，因为这些问题在这两种应用领域中都存在。但是，不论这些应用程序在什么地方运行，本书会聚焦于 Web 应用程序的安全性。你在本书中读到的内容不仅会涉及最新的智能手机，而且会涉及老旧的台式计算机等各种设备。本书会将内容分解为以下几个部分，每个部分都会协助你在安全性建设的道路上走好相应的一步。

- *第一部分*

  无计划不成事。但在计算机行业中，一些最严重灾难的发生恰恰是由于糟糕的计划，而不是没有计划。这一部分会帮助你为公司创建良好的安全计划，即一个考虑到所有最新的用户设备和用户需求的计划。此外还将讨论第三方支持的必要性，因为我们不得不面对这样的事实：在复杂的环境下确实很难独自创建出安全的应用环境。这些内容有助于定位正确的第三方支持，并确保你能收获自己想要的价值。

- *第二部分*

  如今的应用程序开发都会依赖第三方代码库、API 和微服务。这一部分将有助于你思考编码问题。你不会读到太多涉及位或字节层面的内容，但会找到将这些元素成功集成到应用程序中的一些技巧。这一部分会帮助你驾驭应用程序，而不是被它们所驾驭。

- *第三部分*

  测试应用程序安全性的方法有很多种。比如，你可以创建自己的测试套件或者使用其他人创建的套件。第三方程序也能为你做测试。也许你想要知道怎样才能最好地整合不同的测试策略，以确保整个应用程序被完整覆盖。这一部分可以回答你所有关于现代化测试策略的问题，并介绍如何让工作更有效率。

- *第四部分*

  应用程序在某个时间点被发布到生产环境中，并且运行良好。一些应用程序会以这样的方式持续运行多年而不需要适当的维护。不幸的是，现代的应用程序开发需要不断更新，因为黑客在不停地想出新的策略来入侵系统。而你所使用的所有第三方库、API 和微服务的更新则让情况变得更加混乱。这一部分将为你提供一张走出"更新迷宫"的地图，以便应用程序的各部分能够按照你最初的设想保持正常运行。

- *第五部分*

  安全威胁在持续地演变，这意味着你需要一些方法来持续跟进。第一种方法是跟踪这些安全威胁。当然，如果你跟踪每一种威胁，那将一事无成。这一部分描述了你能用来避免信息泛滥的一些技巧。第二种方法是接受额外的培训。事实上，整个公司都需要一些培训来了解最新的安全问题和处理它们的技术。这一部分还以一种所有公司都能采用的方式探讨了安全培训的要求，即使是只有一个人的公司或者创业公司也同样适用。

阅读须知

本书的读者可以是拥有任何头衔的人，比如网页设计师、前端开发人员、UI 设计师、用户体验设计师、交互设计师、美术总监、内容策略师、开发运营人员、产品经理、搜索引擎优化专家、数据科学家、软件工程师或者计算机科学家。大家都有一个共同的需求，即创建安全的 Web 应用程序，让用户能以有意义的方式与之交互。这些人员都是之前开发过 Web 应用程序的专业人士，可能真正需要的是重新了解新形势下的安全技能。这种新形势指的是大部分应用程序是被非传统的方式攻破的，比如通过污染第三方的 API 和库。

本书会给出系统性的安全方案，但不会过多地进行手把手的指导。本书假设你想要了解最新的关于如何在不同级别阻止安全威胁的信息，其中包括对这些威胁的精确解读，以及黑客如何使用它们破坏你的安全措施。

本书包含一些安全编程的示例。要想使用这些例子，你需要具备良好的 CSS3、HTML5 和 JavaScript 编程技术方面的基础。但如果你不具备相关的技能，则可以跳过编程示例，但仍然能够从本书中获得大量有用的信息。编程示例提供只有程序员才会关注的细节。

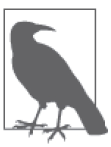
除了编程技巧，更重要的是你接受过一定程度的安全培训。举个例子，如果你根本不知道什么是中间人攻击，那么确实需要先阅读较为基础的图书。本书当然不会假设你是了解中间人攻击各种细节的专家，但确实会认为你之前接触过它。

开发环境

运行本书中的编程示例，只需要文本编辑器和浏览器。文本编辑器必须输出纯文本，不要有任何形式的格式化。它还要能够用正确的文件扩展名（.html、.css 和 .js）来保存这些示例文件。我和多位图书编辑、试读者使用 Linux、Mac 和 Windows 平台上最流行的浏览器测试过书中的例子。事实上，我们甚至在 Windows 10 系统的 Edge 浏览器上测试过这些例子。

本书使用的图标

不同的图标用于传达不同种类的重点信息。本书使用的图标很少，但你需要知道其中每一个图标的含义。



此图标强调一些重要内容，这些内容略微偏离主题或者可能会破坏文字的连贯性。你需要阅读这些说明，因为它们通常会提供有助于你更好执行安全任务的信息。它们还会方便你找到所记得的重要内容，否则你可能很难找到。



此图标表示你必须知道的信息；如果不知道这些信息，你可能会遭受可怕的后果。与说明图标一样，这个图标强调了特殊的内容，这些内容会告诉你可能导致更严重问题的潜在问题。如果你读过某章后没有任何收获，那么一定要将这些警告内容学好并记牢，这能让你在今后避免代价高昂的错误。

补充信息

这一部分包含一些有用的信息，但你在开发 Web 应用程序时不一定需要了解它们。你应该找个时间将这些内容都看一遍，因为它们非常有趣，但没必要立即阅读。这部分内容是对当前话题的补充，不一定是关于该话题的内容。

排版约定

本书使用了下列排版约定。

- *楷体*
  表示新术语或重点强调的内容。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 加粗等宽字体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。

- 等宽斜体（*`Constant width italic`*）
  表示应该由用户输入的值或根据上下文确定的值替换的文本。

获取更多信息

我想要尽量确保你能够获得最好的阅读体验。如果有任何关于本书的问题，请一定发邮件到 John@JohnMuellerBooks.com。你也可以关注本书的博客 http://blog.johnmuellerbooks.com/category/technical/security-for-web-developers/。这个博客会提供更多的内容，并回答读者常问的问题。如果本书有勘误之处，你也能在博客上找到相关的修正。

使用代码示例

补充材料（代码示例、练习等）可以从 https://github.com/oreillymedia/Security_for_Web_Developers 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*Security for Web Developers* by John Paul Mueller (O'Reilly). Copyright 2016 John Paul Mueller, 978-1-49192-864-6."

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online

Safari Books Online（http://www.safaribooksonline.com）是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O’Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

http://shop.oreilly.com/product/0636920041429.do

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：

http://facebook.com/oreilly

请关注我们的 Twitter 动态：

http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：

http://www.youtube.com/oreillymedia

致谢

感谢我的妻子 Rebecca。虽然她已离世，但她的精神存在于我写的每一本书以及每一个词里。当没有人相信我的时候，她一直相信我。

感谢 Russ Mullen、Billy Rios 和 Wade Woolwine 对本书进行的技术编辑。这三位技术编辑极大地提升了书中内容的准确性和深度。很多时候，我都能与他们就书中核心话题的研究进行交流，并寻求他们的帮助。

感谢我的经纪人 Matt Wagner 帮我取得了这份合同，并帮忙打理大多数作者都不会关注的各种细枝末节。我一直感激他的帮助。知道有人愿意帮忙真的是一件非常棒的事情。

许多人阅读了全书或者部分内容，帮助我改进了方法，测试脚本，并提供了所有读者都希望包含的大量输入数据。这些不收酬劳的志愿者以各种各样的方式提供帮助，这里就不一一罗列了。我特别感谢 Eva Beattie、Glenn A. Russell 和 Luca Massaron 的帮助，他们提

供了大量输入数据并读完整本书，忘我地参与到这个项目中来。

最后，我要感谢 Meg Foley、Nicole Shelby、Jasmine Kwityn 以及参与编辑和印制工作的所有其他人员。

电子书

扫描如下二维码，即可购买本书电子版。

第一部分

制订安全计划

在这一部分中，你将了解如何制订用于应用程序开发的安全计划。拥有一份好的安全计划能确保应用达到特定的目标，并且有助于与开发团队商讨如何实现安全性。缺乏良好的安全计划，黑客通常就能轻易地侵入应用程序并给公司制造各种问题。第 1 章将通过让你了解安全计划的组成部分来开启你的安全之旅。

很重要的一点是，你要认识到安全计划的制订不仅应考虑应用程序本身，而且要考虑用户如何使用应用。正如第 2 章中所说的，每一个成功的应用程序都将用户牢记在心头。

此外，你需要明白，现在的应用程序不再处于真空状态，它们会与在线数据源进行交互并依赖第三方代码。明白了这一点后，你还必须思考第三方解决方案会如何影响你的应用，积极影响和消极影响都应考虑在内。使用第三方解决方案能极大地减少你的编码时间。第 3 章会帮你达成这个目标。

第 1 章

定义应用环境

对于任何公司来说，数据都是最重要的资源。可以说，除了数据，公司的任何部分都可以被取代。当数据被篡改、破坏、窃取或删除时，公司会遭受严重的损失。事实上，如果某公司的数据发生了足够多的错误，那么该公司很可能会因此而不复存在。所以，谈到安全的时候，焦点不在于黑客、应用程序、网络或其他任何别人可能告诉过你的东西，而在于数据。因此，这本书是关于数据安全的，其中包括了许多其他话题，但当阅读这些话题时，你需要明白你真正要保护的是什么。

不幸的是，数据孤独地呆在黑暗中时并没有多少用处。无论你的服务器有多高档，或者你的数据库有多强大的存储能力，直到你利用数据时，数据才会有价值。我们需要使用应用程序来管理数据，这就是本章要讨论应用环境的原因。

然而，在进一步阅读之前，很重要的一件事是，要先明确应用和数据是如何交互的，因为如果对此没有了解，那本章余下的内容对你不会有太大用处。一个应用程序无论多么复杂，它只会对数据进行四种操作。你可以用 CRUD 这个缩写来指代这四种操作。

- 创建（Create）
- 读取（Read）
- 更新（Update）
- 删除（Delete）

接下来会讨论与 Web 环境有关的数据、应用程序和 CRUD。你会发现数据安全是如何影响 Web 开发中这三个方面的。记住，虽然数据是重点，但所需的 CRUD 任务是由应用程序来执行的。保证数据安全意味着要理解应用程序的运行环境，进而理解应用程序管理的数据所面临的威胁。

1.1 明确Web应用威胁

你可以在互联网上找到 Web 应用威胁的清单。其中有一些是全面而不带偏见的，有一些论述了作者认为的最重要的威胁，还有一些会告诉你哪些是最常见的威胁，此外你还可以找到各种各样的其他清单。所有这些清单的问题在于，它们的作者并不了解你的应用程序。比如，SQL 注入攻击只在应用程序以某种方式使用 SQL 时才会成功，但你的应用程序可能并未采用这种方式。

很明显，你需要知道在哪些场景下应该检查什么，而这些清单确实是个好的起点。然而，你需要根据自己的应用程序来考虑这些清单的内容。此外，不要只依赖一份清单，尽量参考多份，这样你能对可能的威胁有更好的认识。带着这样的需求，下面列出了现在最常见的一些 Web 应用威胁。

- *缓冲区溢出*
  攻击者试图将足够多的数据发送到输入缓冲区中，以便让应用程序或输出缓冲区溢出，从而导致在缓冲区外的内存数据被损坏。某些形式的缓冲区溢出能使攻击者执行看起来不可能完成的任务，因为受影响的内存中保存着可执行的代码。解决这种问题最好的办法是，对应用程序处理的所有数据都执行边界和大小检查，无论是输入还是输出。你可以在网页 http://www.upenn.edu/computing/security/swat/SWAT_Top_Ten_A5.php 和 https://www.owasp.org/index.php/Buffer_Overflows 上看到更多关于 Web 应用程序缓冲区溢出的内容。

- *代码注入*
  一个实体以中间人攻击（man-in-the-middle-attack）的形式将代码添加到服务器和客户端（如浏览器）之间的数据流中。被攻击方经常会将这种注入代码视为原始页面的一部分，但它可能包含任何东西。当然，被攻击方甚至看不到这些注入的代码。它可能潜伏在后台，随时准备给应用程序制造各种各样的问题。解决该攻击的一种好方法是确保使用了加密的数据流、HTTPS 协议和代码验证（如果可能）。提供客户端反馈机制也是一个好主意。

> 代码注入往往发生得比你想象的频繁。在某些情况下，代码注入甚至不算攻击的一部分，虽然也可能是。最近有一篇文章（http://www.infoworld.com/article/2925839/net-neutrality/code-injection-new-low-isps.html）讨论了互联网服务提供商（Internet service provider，ISP）如何将 JavaScript 代码注入到数据流中以便在页面上放置悬浮广告。而为了确定要投放哪一类广告，ISP 还会监听数据传输。

- *跨站脚本*（cross-site scripting，XSS）
  攻击者将 JavaScript 脚本或其他可执行代码注入到应用程序的输出流中。接收者会将你的应用程序看作感染源，但其实它并不是。大多数时候，不应该在没有严格验证的情况下允许用户通过你的应用程序直接将数据发送给其他用户。对于像博客这样的应用程序来说，采取审核机制是很有必要的，以此确保你的应用程序最终不会沦为病毒的宿主，或者出现更糟糕的情况：被掺入看似无害的数据。

> 很少有专家会提醒你检查输出数据。但是，你并不能准确地知道自己的应用程序是否值得信赖。黑客有可能破解应用程序并污染输出数据。验证核查应涵盖输入数据和输出数据。

- *文件上传*
  即使看上去是无害的，但每一次的文件上传也应该被认为是可疑的。黑客有时候会利用服务器的文件上传功能来上传后门程序，这些文件会包含一些很危险的东西。如果可能，应禁止服务器的文件上传。当然，我们通常无法提供这种级别的安全性，所以你需要限制允许上传的文件类型，然后扫描这些文件，检查它们是否有问题。尽可能验证文件始终是一个好主意。例如，一些文件会在头部包含一段签名信息，你可以用它来判断这个文件的合法性。不要单纯依赖文件扩展名进行排除，黑客经常会将一个文件伪装成另一种类型来绕过服务器的安全验证。

- *硬编码的认证*
  出于测试目的，开发人员经常将认证信息放在应用程序的初始化文件中。我们需要去掉这些硬编码的认证信息，以集中式存储的安全信息替代。将数据存储在安全的位置而不是 Web 应用服务器上是很重要的，这样黑客就不能轻易地查看到可用于访问应用程序的凭据。如果你的应用程序确实需要一些初始化文件，那么要确保这些文件放在 Web 根目录以外的地方，以防止黑客偶然发现它们。

- *发现隐藏或受限制的文件 / 目录*
  当应用程序允许输入一些特殊字符（比如斜杠或反斜杠）时，黑客就有可能发现隐藏或受限制的文件和目录。这些地方可能包含各种有助于黑客攻击你的系统的信息。尽可能禁止使用特殊字符是一个很好的主意。除此之外，请把重要的文件放在 Web 根目录之外且操作系统能直接控制的地方。

- *缺少认证或者认证不正确*
  知道谁正在使用应用程序是很重要的，特别是在你处理敏感数据的时候。许多 Web 应用程序依赖公共账户来执行某些任务，这意味着不可能知道是谁访问了这个账户。应该避免因为任何原因而使用访客账户，并为每一个用户分配独立的账户。

- *缺少授权或者授权不正确*
  即使知道了谁正在使用你的应用，还有一件重要的事：只为用户提供执行特定任务时所需要的权限。此外，授权应该反映用户的访问方式，比如一个桌面系统通过本地网络访问应用程序可能比一部智能手机在当地咖啡店里访问应用程序更安全。依靠安全提升来协助处理敏感任务，能够让你在处理任务之外的时间里维持最低的授权。任何降低用户权限的行为都有助于维持一个安全的环境。

- *缺少加密或者加密不正确*
  使用加密技术在两个终端之间传输数据可以防止黑客监听通信。重点是要跟进最新的加密技术，并且依靠用户环境所能支持的最好加密算法。例如，三重加密算法（Triple Data Encryption，3DES）已经不再安全了，但一些组织仍在使用它。高级加密标准（Advanced Encryption Standard，AES）目前仍然是安全的，但你要尽可能用最长的密钥

来加大破解难度。

- *操作系统命令注入*
  攻击者会修改应用程序使用的操作系统命令来执行特定的任务。你的 Web 应用程序一开始可能就不应该使用系统调用。但是，如果必须使用，请确保你的应用程序在一个沙盒环境中运行。

> 有些专家会强调在某些使用场景中需要校验输入数据，但对其他的使用场景不作此要求。请始终对从任意数据源接收到的任何数据进行校验。你无法知道黑客会用什么工具来侵入系统或以怎样的方式来制造损害。输入数据总是可疑的，即使该数据来自你自己的服务器。当你处理安全相关的任务时，变得多疑是件好事。

- *参数篡改*
  黑客会对作为请求头或 URL 一部分的参数进行尝试。例如，使用 Google 搜索时，你可以通过改变 URL 来改变搜索结果。请确保加密你在浏览器和服务器之间传递的任何参数。此外，在传递参数时，使用安全的网络传输协议，比如 HTTPS。

> 如果有足够的时间和精力，黑客仍然能操纵参数。仔细定义好参数的取值范围和数据类型也很重要，这能减少此种攻击带来的潜在问题。

- *远程代码包含*
  如今的很多 Web 应用程序依赖包含外部库、框架和 API。大多数情况下，包含语句会含有一个相对路径或者使用一个记录着硬编码路径的变量，这样做是为了方便以后更换远程代码的位置。当黑客能够获取到这些路径信息并更改它们的时候，远程代码包含就可能被指向黑客想要包含的任何代码，从而使得黑客可以完全掌控你的系统。避免这种问题的最佳方法就是尽量使用硬编码的完整路径，尽管这会使代码维护变得困难。

> 许多专家会建议你使用经过审核的库和框架来执行有风险的任务。但是，这些添加的插件不过是更多的代码而已。黑客总能发现破坏和规避这些库以及框架的方法。你仍然需要确保应用程序和所有它依赖的代码能够安全地与外界交互，这意味着需要有额外的测试。使用外部库和框架确实能降低维护成本并且能提供及时的 bug 修复，但是 bug 仍旧会存在并且你仍需要保持警惕。保证数据安全没有万全之策。第 6 章包含了更多关于使用外部库和框架的内容。

- *会话劫持*
  每次用户登录到你的 Web 服务器，服务器就会为其创建一个唯一的会话。会话劫持者会侵入到会话中并拦截用户与服务器之间传输的数据。含有劫持会话所需信息的三个最常见的地方是：cookie、URL 重写和隐藏域。黑客会在这些地方查找会话信息。只要保

持会话信息是加密的，你就能降低会话被劫持的风险。例如，确保使用 HTTPS 协议来登录。你也应该避免使用可被预测的会话 ID。

- SQL 注入
  攻击者会修改由应用程序创建的查询，而这个查询是由用户或其他输入引起的。大部分 SQL 注入的情况是，应用程序需要的是用于查询的数据，但它实际接收到的却是一些 SQL 片段。其他形式的 SQL 注入攻击与使用了转义字符或其他意想不到的字符或字符串有关。一个避免 SQL 注入攻击的好办法是避免动态查询。

看起来我们周围存在着许多不同的网络威胁，但如果在网上搜索的时间足够长，你会轻易地将这份清单的长度变成现在的 3 倍，然而这些也只是众多黑客攻击手段中的一小部分。当继续读这本书时，你会碰到更多种类的网络威胁并开始了解解决它们的方法。不用担心，大部分情况的解决方案都是一些常识，并且一个方案能解决不止一个问题。例如，再看看上面的清单，你会发现仅仅使用 HTTPS 就可以解决其中的多个问题。

考虑隐私安全问题

当一家公司涉足安全时，它往往会首先关注自己的数据安全，毕竟，如果公司的数据丢失、被损坏、被修改或以别的方式变得不可用，那它可能会面临破产。公司下一个级别的安全考虑通常涉及第三方，比如合作伙伴。用户数据的安全性通常是最后才考虑的，并且，很多公司根本不考虑客户数据的安全。问题是，许多用户和客户把他们的数据安全看得非常重要。所有的隐私问题都归结于用户数据的保护，在没有经过用户同意或用户不知情的情况下，不能滥用或暴露用户信息。总之，在构建应用程序时，必须把用户数据的隐私作为一个重要的安全问题纳入考虑范围。

最近的一篇文章（http://www.infoworld.com/article/2925292/internet-privacy/feds-vs-silicon-valley-who-do-you-trust-less.html）指出，用户和客户把科技行业视为糟糕的数据受托方。事实上，科技行业确实在这方面已落后于政府，人们相信政府会更好地保护他们的信息。许多科技公司公开支持其他实体（比如政府）提出的增强安全性的策略，但私底下却用各种方式侵犯用户和客户的隐私。这种两面派的做法比起科技行业公开承认侵犯用户和客户的数据更糟糕。

要创建一个真正安全的应用程序，必须涵盖安全的每一个方面，包括用户和客户的数据。这要求应用程序只获取和管理其执行相关任务所需的数据，而不去涉及那些不再需要的数据。只有对操作的所有数据都遵循相同的原则，不管它们从何而来，应用程序才会收获信任。

1.2 理解软件安全保障

软件的目的是与数据进行交互。但是，软件本身也是一种数据。事实上，数据有很多你可能从未考虑过的形式，并且其影响比你想象的要广泛得多。随着物联网（Internet of Things，IoT）时代的到来，如今的数据可能具有抽象和物理双重影响力，这在几年前甚至无法想象。成功入侵应用程序之后，黑客可以做破坏电网和向自来水系统投毒这样的事

情。从个人层面来说，黑客同样可以悄悄地把你家的温度调高到可怕的程度，关掉你家所有的灯，或者通过你的网络摄像头监视你，这只是一些例子。软件安全保障（Software Security Assurance，SSA）的观点是，软件需要某种监管来确保它不会导致其使用、控制和保护的数据及资源发生丢失、误差、篡改、不可用或误用。这一要求是 SSA 的一部分。接下来会更详细地讨论 SSA。

SSA 目前并不是一个实际的标准。它是许多公司基于自身需求进行量化并写入文档的一个概念。在这许多的文档中出现了一些相同的基本模式，进而 SSA 这一术语用以指代这些确保软件安全的做法。通过查阅各大公司的在线 SSA 文档，你会发现 SSA 是如何影响这些公司的，比如 Oracle（https://www.oracle.com/support/assurance/index.html）和 Microsoft（https://msdn.microsoft.com/library/windows/desktop/84aed186-1d75-4366-8e61-8d258746bopq.aspx）。事实上，目前许多大公司都在适当的地方有某些形式的 SSA。

1.2.1 考虑OSSAP

为了使 SSA 在 Web 应用程序中成为现实，有一个重要的网站需要关注，那就是开放 Web 应用安全项目（Open Web Application Security Project，OWASP，https://www.owasp.org/index.php/OWASP_Software_Security_Assurance_Process），如图 1-1 所示。这个网站对如何使 OWASP 的安全软件保障流程（OWASP Security Software Assurance Process，OSSAP）成为软件开发生命周期（Software Development Lifecycle，SDLC）的一部分进行了流程分解。是的，这里有一大堆首字母缩写，不过你需要了解这个组织，这会帮你为自己的应用程序创建一套流程来与其他公司所做的工作相匹配。此外，这个网站上的信息能帮助你为应用程序开发一套真正能用的安全流程，使其成为开发流程中的一部分，而这并不会花费太多的时间。

虽然 OSSAP 提供了一个很棒的框架来确保应用满足 SSA 的要求，但你并不需要与该组织有任何的互动。这个组织已经将它的方法授权给 SSA。不过，这个组织现在还在发展中，你可以在网站上发现很多待定的事项，这些都会在后面的时间里补充上。当然，你需要一个当前能用的计划，所以 OWASP 和它的 OSSAP 为你提供了研究当前解决方案的平台，并且在今后也能获得更多的帮助。

在应用程序中将 SSA 作为软件开发生命周期一部分的根本原因，是想尽可能确保软件是可靠和无差错的。在与某些人的交谈中，会发现他们认为 SSA 会解决你可能遇到的所有潜在安全问题，可实际上并非如此。SSA 会提升软件质量，但没有一款软件是没有差错的。假如你确实开发了一款没有错误的软件，仍然需要考虑用户、环境、网络和所有其他的软件安全问题。因此，SSA 只是安全版图中的一小块，而实现 SSA 也只能解决许多而不是全部的安全问题。最佳的做法就是将安全建设视为一个持续的过程。

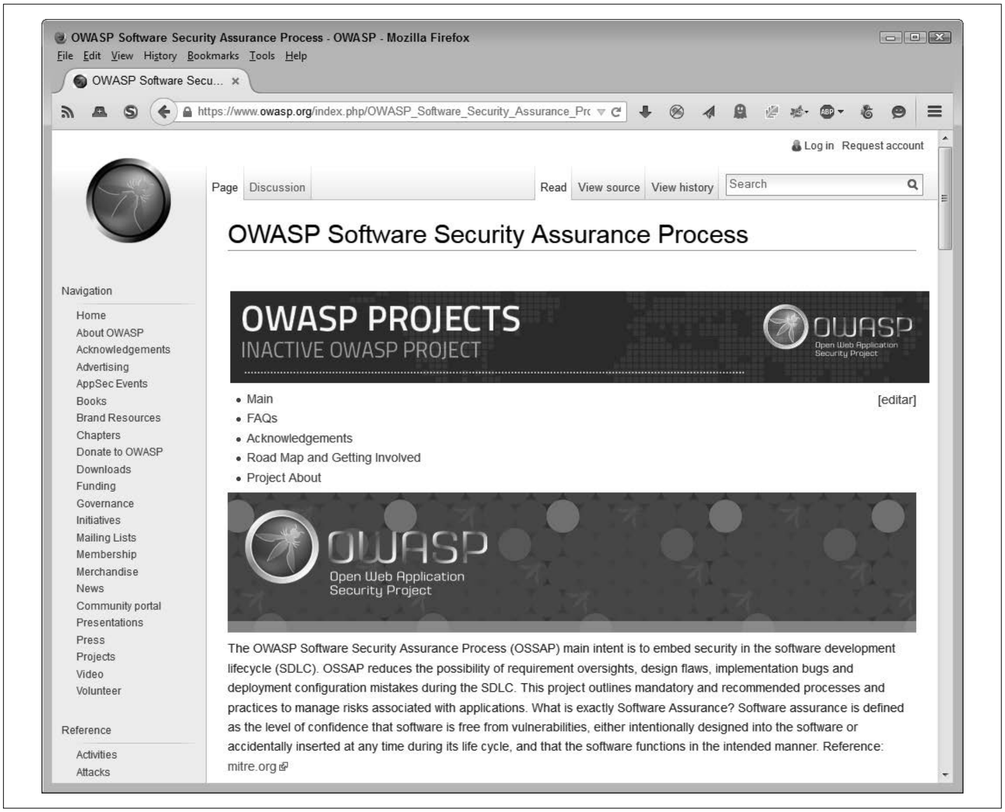


图 1-1：OWASP 网站会告诉你关于 Web 应用的 SSA

1.2.2　定义SSA的要求

要将实现 SSA 作为应用程序的一部分，第一步需要先定义 SSA 的要求。这些要求会帮你确定软件的当前状态、需要解决的问题，以及这些问题的严重程度。在定义问题之后，你能够明确修复进程以及任何其他要求，以确保软件安全。实际上，你可以将 SSA 分解为以下 8 个步骤。

(1) 评估软件和制订修复计划。
(2) 定义数据的各种安全风险并进行分类，优先修复最严重的风险。
(3) 执行全面的代码检查。
(4) 进行必要的更改。
(5) 测试修复并在生产环境中验证它们确实是有效的。
(6) 制定防御机制来保护应用程序的访问及其管理的数据。
(7) 衡量你所做的这些更改的有效性。
(8) 以适当的方式培训管理者、用户和开发人员，以确保良好的应用安全性。

> **另一种 SSA 策略**
>
> 当涉及安全问题的时候，你会发现针对一个问题有许多解决方法。根据公司的文化和工作方式，你也许会发现有另一种技巧可以确保 SSA 工作得更好。一些安全专家会建议以下这些步骤。
>
> (1) 把安全性要求置于产品需求之上（这一条只需要在启动新软件开发时执行）。
> (2) 定义并与开发人员沟通写代码时必须遵守的安全编码规范。
> (3) 开发人员在写新代码时要启用自动代码检查。
> (4) 完成应用开发之后要执行全面的代码检查。
> (5) 对整个应用程序进行渗透测试和脆弱性评估。
> (6) 评估测试结果以找到安全和业务之间的平衡点。制订计划，修复已确定的、对业务有很大危害的漏洞。
> (7) 执行所有需要进行的安全性修复。
> (8) 重复步骤 (5)。

1.2.3 对数据和资源分类

这一过程包括识别应用程序涉及的各种数据，包括它自身的代码和配置信息。一旦识别了所有数据，你就要对其进行分类来确定所需的安全级别，从而保护这些数据。数据有很多分类方式，而用什么方式则取决于公司的需要和对数据的定位。例如，有些数据可能只会给公司带来麻烦，而有些数据则可能对人造成伤害。定义数据安全漏洞会如何影响整个安全环境，是至关重要的。

完成数据分类之后，就可以开始用这些分类信息来执行各种任务。例如，你可以考虑通过以下方式减少漏洞：

- 制定编码规范
- 执行强制的开发人员培训
- 在开发团队中聘请安全主管
- 使用专门定位安全问题的自动化测试程序

所有这些方法都指向公司使用和依赖的资源，以确保应用程序正确地管理数据。资源分类意味着确定对特定资源的重视程度。例如，拒绝培训开发人员会比拒绝培训用户产生更大的影响，因为开发人员与应用程序是一体的。当然，培训对所有人都很重要。既然这样，将资源分类会让你明确在何地如何花钱才能获得最佳的投资回报，同时仍然能满足应用的安全性目标。

1.2.4 进行必要的分析

作为 SSA 的一部分，你需要对应用程序进行分析（包括威胁建模、用户界面漏洞和数据展示漏洞）。准确地知道代码可能包含怎样的缺陷是很重要的。这里的关键词是可能。在进行深入分析之前，你无法知道代码中实际的安全问题。Web 应用程序特别容易有潜藏的

问题，因为与桌面端应用程序不同，其代码会出现在许多地方，并且脚本语言易于隐藏问题，而经过编译的应用程序则没有这种情况，因为脚本语言是在运行时而不是编译时进行解析的。

> 重要的一点是，要理解数据安全不仅与代码有关，还涉及创建代码所需的工具和使用这些工具的开发人员的技能。当一家公司选择错误的工具来完成这个工作时，出现安全漏洞的风险会大大提高，因为这些工具可能无法创建出严格符合预期的代码。同样，当使用工具的开发人员不具备必需的技能时，软件出现安全漏洞就毫不奇怪了，但更熟练的开发人员能避免这些漏洞。
>
> 某些专家宣称，有些公司实际上容忍不合格的工作。在大多数情况下，给这种行为找的借口是开发进度落后于预期或缺少必需的工具或专业技能。公司有可能会专门设计用来解决安全问题的软件（比如防火墙），这并不能减轻开发人员开发安全代码的重任。公司需要坚持编码标准来确保取得良好的结果。

1. 逻辑

与应用程序及其管理的数据进行交互是一系列过程。虽然用户可能会以看起来随机的方式执行任务，但具体的任务总会遵循一定的模式，因为用户必须遵循一定的过程逻辑来获得良好的结果。通过记录及理解这些过程，你能够从实际的角度分析应用程序的逻辑。用户依赖于由开发人员设计应用程序的方式决定的特定过程。改变这个设计势必会引起过程的改变。

分析的目的是要找出过程逻辑中的安全漏洞。例如，应用程序可能会允许用户保持其登录状态，却没有对过期后的登录状态有效性进行检测。这里的问题是，在线用户可能根本就不在线，而是其他人利用了他的凭据登录了系统，并且没人知道这件事，因为所有人都觉得这个用户是像平时一样正常登录该系统。

但是，对数据漏洞的分析可采用其他的方式。例如，一个序列号可能由几个可量化的元素构成。为了得到好的序列号，应用程序可获取这些元素并根据这些元素来构建序列号，而不是把序列号当作一个整体。这样做是为了让过程逻辑更简洁、清晰和更不易出错，从而使数据库不会存储大量不良信息。

2. 数据

也许看起来不可能从安全性的角度对数据进行太多的分析，但确实有很多问题需要予以考虑。事实上，很多公司的数据分析都做得不够好，因为他们把重点放在了如何管理和使用数据上，而忽略了如何确保数据的安全（有理由相信你需要解决这三个问题）。当分析数据时，你必须考虑以下这些问题：

- 谁能访问数据
- 数据以什么格式存储
- 数据何时可被访问
- 数据存储在什么地方
- 为什么每个数据项都能作为应用程序的一部分被使用
- 数据如何分解成各个部分，以及组合数据供应用程序使用的后果

比如，一些应用程序没有执行数据隐藏，但其实数据隐藏应该是任何好的应用程序应该有的重要特征。它是指只为用户提供其执行给定任务时确实需要的信息。

一些应用程序对数据格式的处理也不正确。比如，存储明文密码，只要有人攻破了系统，几乎可以肯定会出问题。更好的做法是存储经过安全加密算法（还没有被破解的算法）处理的密码散列值。散列值对于攻破系统的人来说毫无用处，因为应用程序需要的是用来生成散列值的原密码。

让所有数据一直可访问也不是好的做法。对于敏感数据来说，应当只在有人监管其使用并能对用户的异常行为作出及时反应的前提下，才将其显示出来。

把敏感数据存储在云端是相当糟糕的做法。是的，采用云存储能更方便和快捷地访问数据，但也会使数据更容易遭受攻击。当你能直接使用保证数据安全的所有安全特性时，请将敏感数据保存在本地服务器上。

应用开发人员喜欢提供过多的信息。你应该采用数据隐藏将面向管理员的数据与其他人员隔绝。然而，有些数据在应用程序中根本没有被任何一处地方使用。如果在处理一个任务时确实不需要用到某些数据，那就不要将这些数据添加到应用中。

如今的许多数据项是其他数据元素的聚合。通过探测数据聚合方式分解数据项来发现数据的各个组成部分，黑客有可能窃取到关于公司的大量信息。关键是要考虑如何将数据合并，并添加安全防护措施，从而让数据源更难被发现。

3. 界面

现今软件的一大问题是包含了不必要的功能。一个应用程序应该满足一组特定的目标，或者执行一组特定的任务。总有些人在想，如果软件能有一些额外的特性，可能会更好，虽然这些特性与其要达成的核心目标完全无关。*功能膨胀*这个词由来已久，你经常在涉及金钱消耗的场景中发现它，比如应用程序性能问题的根源、用户培训费用的提升，以及开发进度的延迟等。但是，受功能膨胀的影响，应用程序界面的问题会因攻击面（attack surface）的增加而对安全性产生重大影响。攻击面的每一次增加都会为黑客提供更多入侵公司系统的机会。将不必要的功能移除或将它们移到另一个不同的应用中，可以减少应用的攻击面，从而让应用程序更安全。当然，你也会省下不少钱。

另一个潜在的威胁是提示界面，它提供给潜在的黑客太多信息或功能，从而丢失了应用程序的安全性。虽然提供一种途径让用户找回密码是有必要的，但有些实现方式确实有可能让黑客得到用户的密码从而伪装成该用户。黑客甚至可能会修改用户的密码，使得真正的用户无法登录（但是这一举动会适得其反，因为管理员能轻松恢复用户的访问权限）。一个更好的系统要确保用户在干任何其他事情之前先发送认证请求，并确保管理员能以安全的方式发送登录信息。

4. 约束

约束就是一种方法，在允许执行某一行为之前，用来确保其满足特定的标准。例如，除非用户拥有相关权限，否则禁止其访问数据元素就是一种约束。但是，约束有其他更重要的形式。最重要的约束是确定用户如何管理数据。大部分用户只需要读取数据，但应用程序通常提供了读 / 写访问，从而打开了一个巨大的安全缺口。

对于数据也需要考虑约束。当处理数据时，你必须精确定义用什么来保证数据的唯一性，并确保应用程序不会破坏任何与唯一性有关的规则。有了这个想法，你一般需要考虑下面几种约束：

- 确保数据有正确的类型
- 定义数据能接受的值的范围
- 明确数据长度的最大值和最小值
- 列出所有不能接受的数据值

1.3 探究与语言相关的问题

应用环境是由所采用的开发语言定义的。正如每一种语言都有善于处理某些任务的功能，每一种语言也都有造成安全风险的潜在问题。即使是低级语言，尽管具有灵活性，也有因复杂性引起的问题。当然，基于 Web 的前端应用程序通常依赖三种特定的语言：HTML、CSS 和 JavaScript。接下来将描述与这三种语言相关的问题。

1.3.1 定义HTML的关键问题

HTML5 因支持极为广泛的平台而变得相当热门。不需要开发人员编写特殊的代码，同一个应用程序就能很好地在用户的桌面电脑、平板设备和智能手机上运行。通常，库、API 和微服务不需要开发人员的介入就能提供自动适配宿主系统的内容。但是，HTML5 提供的这种灵活性也存在问题。以下清单列出了你在使用 HTML5 时会遇到的一些关键的安全性问题。

- 代码注入

  HTML5 里有大量黑客可用来注入恶意代码的方法，包括你通常不认为可疑的数据源，比如一段 YouTube 视频或是流式音乐。

- 用户跟踪

  因为大多数情况下应用程序使用了不同来源的代码，所以你可能会发现库、API 或微服务实际上执行了一些类型的用户跟踪，而黑客能利用其来获取公司信息。你给黑客的每一份信息都会让攻克系统安全变得更容易。

- 污染输入

  除非你提供自己的输入项检查，否则 HTML5 会放行所有用户想要提供的输入。你可能只需要一个数字型的值，但用户可能输入一段脚本代码。尝试对输入进行彻底检测以确保真正得到你要求的输入数据，这在客户端近乎是不可能的，所以你还需要确保有强大的服务器端检查机制。

HTML5 的攻击面

HTML5 增加了许多有趣的、黑客非常喜欢的特性，因为这些特性提供了更多的攻击面。例如，在你发现恶意软件能轻易利用 LocalStorage 入侵系统的本地存储之前，LocalStorage

看起来是个很好的东西。你能在网页 http://www.slideshare.net/shreeraj/html5-localstorage-attack-vectors 和 https://blog.whitehatsec.com/web-storage-security/ 上读到这种攻击的相关细节（包括一些攻击案例）。

另一个有问题的特性是 WebSocket。这一特性允许客户端与服务器端的双向通信。唯一的问题是，客户端可能在运行不可靠的代码。你能在网页 http://www.slideshare.net/SergeyShekyan/bay-threat-2012-websockets 和 http://blog.kotowicz.net/2011/03/html5-websockets-security-new-tool-for.html 上看到更多关于这一问题的资料，其中有些例子会告诉你如何使用 WebSocket 来克服安全保护的问题以及潜在的解决方案。

1.3.2 定义CSS的关键问题

Web 应用程序非常依赖 CSS3 来创建美观的界面，而不需要为每个设备硬编码相关信息。已有的 CSS3 代码库使得创建专业美观的应用变得简单，并且用户能更换它们来满足不同的需求。例如，用户可能需要在某一设备上有一个不同的界面，或者需要一种特殊格式的展现来满足一个特定的需求。以下列举了一些在使用 CSS3 时会遇到的关键安全性问题。

- 设计主导
  CSS3 代码导致安全问题的一个主要原因是由设计来主导一切。标准委员会最初设计 CSS 是用来控制 HTML 元素的展现的，而不是影响整个 Web 页面的展现。因此，设计师从来不会考虑相关的安全问题，因为 CSS 就不是用在这一领域的语言。问题在于 CSS 的样式级联部分不允许 CSS3 知道除了其父节点之外的任何信息。因此，黑客能创建出一种声称要做某件事的界面，实际上却做着另外的事情。一些库（如 jQuery）确实能帮你解决这一问题。

- 上传的 CSS
  在某些情况下，应用程序的设计者会允许用户上传一份 CSS 文件来获得某一特定的应用外观或让其在一个特定的平台上运行得更加良好。但是，上传的 CSS 也可能包含一些代码，使得黑客更容易破坏设在各个地方的安全措施，或者在可见范围内隐藏脏东西。例如，黑客会在 CSS 里隐藏 URL，从而将应用重定向到不安全的服务器上。

- CSS 着色器
  一种特殊的 CSS 用法，它允许访问浏览器数据和跨域数据，从而会产生一些极端的问题。本书后面的章节会详细讨论相关的细节，但你能在网页 http://www.w3.org/Graphics/fx/wiki/CSS_Shaders_Security 上快速浏览这一话题。重要的是要明白，有时候在屏幕上展现数据的行为会导致你最初未考虑到的潜在安全漏洞。

1.3.3 定义JavaScript的关键问题

JavaScript 和 HTML5 的结合已经创造了整个 Web 应用的奇迹。没有这两种语言的结合，就不可能创造出在所有设备上都运行良好的各种应用程序。用户甚至不再想用原来的那种应用程序，因为其不能提供这种能力。现在，用户能在任何地方用适合该位置的设备完成工作。但是，JavaScript 是一门有着严重安全缺陷的脚本语言。下面列出了你在使用

JavaScript 时会遇到的一些关键的安全性问题。

- 跨站脚本（XSS）
  这一问题在本章前面出现过，因为它是一个非常严重的问题。只要你在沙盒环境之外运行 JavaScript，黑客就有可能对你的应用程序做一些讨厌的动作。
- 跨站请求伪造（CSRF）
  一段脚本能够利用保存在 cookie 里的用户凭据来访问其他站点。在这些网站上，黑客能执行应用程序设计目的之外的各种任务。例如，黑客能以用户的名义，进行账户篡改、窃取数据、欺诈以及许多其他的违法活动。
- 浏览器及其插件的漏洞
  很多黑客会利用已知的浏览器及浏览器插件的漏洞来强迫应用程序执行其不应执行的任务。例如，用户的系统可能突然变成了向其他系统传送病毒代码的傀儡。黑客能够破坏的程度受限于相关的漏洞。一般来说，你要确保自己安装所有的更新，并要了解这些安全漏洞会如何影响应用程序的作业。

1.4 考虑端点的防御要素

端点是指网络传输的目的地，比如一个服务或一个浏览器。当数据包到达端点时，其包含的数据会被解包出来并提供给应用程序做进一步的处理。端点安全是至关重要的，因为端点是网络的一个主要入口点。除非端点是安全的，否则网络会接收到恶意的数据。此外，糟糕的端点安全性会对网络里的其他节点造成损害。接下来会讨论端点安全的三个阶段：预防、检测和修复。

> 重要的是不要低估端点安全性对应用和网络基础设施的影响。一些端点的情况非常复杂，以至于其后果难以被检测甚至理解。最近的一篇文章（http://www.infoworld.com/article/2926221/security/large-scale-attack-hijacks-routers-through-users-browsers.html）讨论了一种路由器攻击，攻击者将毫无防范的用户定向到一个特定的网站。此类攻击针对的是用户发起域名系统（DNS）请求时依赖的路由器。通过完全控制这种路由器，攻击者能将用户重定向到他控制的站点。

1.4.1 预防安全漏洞

避免安全陷阱的第一步是一开始就承认陷阱是存在的。问题是，如今大多数公司并不认为自己会遭遇数据安全陷阱，认为这总是发生在其他安全性不高的公司。然而，根据 Ponemon 研究所关于 2014 年全球网络犯罪造成损失的统计报告（http://info.hpenterprisesecurity.com/LP_CP_424710_Ponemon_ALL），2014 年网络犯罪导致的损失金额高达 1270 万美元，而 2010 年该值是 650 万美元。显然，所有这些入侵行为不会只发生在别人的公司，它们也很容易发生在你的公司，所以，最好假设某个地方的某个黑客已盯上了你的公司。事实上，如果一开始就意识到黑客不仅会入侵公司系统，还会偷走你的财物，你就会真正为现实生活中的场景做好准备。你构建的所有应用程序必须足够可靠以实现以下功能：

- 对抗普通的攻击
- 在安全措施失效时发出警报
- 避免对可能会发生漏洞的地方做出假设
- 要假设即使是受过训练的用户也会犯导致安全事故的错误

> 不要假设安全漏洞只存在于某些平台。安全漏洞会出现在任何运行非定制软件的平台上。开发人员为某一平台做的准备越少，漏洞就会变得越严重。例如，许多人觉得销售点（point-of-sale，POS）终端机是安全的，但是黑客目前正在大力攻击此类设备（http://www.computerworld.com/article/2925583/security/attackers-use-email-spam-to-infect-pos-terminals.html）以获取信用卡信息。但有趣的是，如果店员正确使用 POS 终端机，那此类攻击就无法奏效。这个例子表明培训和强有力的策略有助于维持系统安全。当然，应用程序本身还是应该足够可靠，以对抗这些攻击。

继续阅读本书，你会发现一些可减少安全漏洞的有用技巧。一旦你承认漏洞会（并且很可能）发生，预防安全漏洞的关键就在于以下几点。

- 创建用户理解并喜欢使用的应用（见第 2 章）
- 谨慎选择外部数据（详见 1.6.4 节）
- 构建具有天然入侵屏障的应用（见第 4 章）
- 测试所写代码的可靠性，并详细记录故障及其原因（见第 5 章）
- 谨慎选择外部库、API 和微服务（详见 1.6 节）
- 对所有的应用元素(甚至是并非你自己所有的元素)执行全面的测试策略(详见第三部分)
- 管理应用程序的各个部分,以确保安全防护措施不会在应用上线后失效（详见第四部分）
- 持续跟进最新的安全威胁与解决策略（见第 16 章）
- 培训开发人员，让他们在每个项目里都自始至终考虑安全问题（见第 17 章）

1.4.2 检测安全漏洞

所有公司最不希望发生的事是，听到别人的安全事故。在行业媒体上读到自己公司无力保护用户数据的报道是开始新一天的最糟糕方式，但这却是许多公司学习安全漏洞的方式。如果一家公司假设已经发生了数据泄露，那它最不可能遭受数据泄露导致的永久性损失，因而最有可能节省成本。公司要能检测数据泄露的出现并防止它变成一个事故，而不应该在它发生之后才耗费时间与资源去修复。检测意味着要提供必需的代码以作为应用程序的一部分，并确保这些检测方法是设计用来处理目前的安全威胁的。

作为一个整体，公司需要一个响应安全漏洞的团队。但是开发团队也需要在适当的地方独立地对安全漏洞进行检测。就目前而言，大部分的开发团队需要以下这些领域的专家：

- 网络
- 数据库管理
- 应用设计与开发
- 移动技术
- 网络取证

- 法规遵从

每个应用程序都需要一个团队，这个团队能够定期讨论与应用程序相关的安全需求和安全威胁。除此之外，审查各种威胁的情况并确定其发生时应该采取的对应措施也很重要。准备好这些之后，你更有可能较早地检测到安全漏洞，即在管理层急匆匆闯进你的办公室要求你给个解释之前。

1.4.3 修复受损的软件

当安全事故确实发生了，公司与之相关的任何团队都必须准备承担责任并采取修复措施。公司层面需要明白，如果不能尽快修复安全漏洞并将系统恢复至事故发生前的状态，那么公司可能面临破产。换言之，即使是公司的高层，你也可能要找一份新的工作了。

安全负责人可能让开发团队帮忙定位攻击者。安全信息与事件管理（security information and event management，SIEM）软件能帮忙查阅指向问题根源的日志。当然，这里假设你的应用程序确实记录了相关的日志。修复过程的一部分是，一开始就要对应用程序构建日志记录与跟踪功能。没有这些信息，要想找到罪魁祸首并阻止其攻击通常注定会失败。

修复过程应该包含对应用程序各部件的更新和补丁进行检查的策略。为了达成这一目标，必须维护良好的应用文档。当安全问题出现时才创建外部资源清单就太迟了，你必须在安全问题发生之前就将外部资源清单掌握在手中。当然，为了确保安全问题不再发生，开发团队需要对所有应用程序需要的更新进行测试。最后，你需要确保数据在整个过程中始终保持安全，并执行应用程序所需的所有数据恢复。

1.5 处理云存储

在一个员工要求随时随地能用身边的任何一种设备访问数据的世界里，云存储是一个必然的灾难。用户有各种各样的云存储解决方案可以选择，但现在最热门的是 Dropbox（https://www.dropbox.com/），到 2014 年年底它已拥有 3 亿多用户。Dropbox（以及很多其他的云存储提供商）有着盛衰无常的安全历史。例如，2011 年，Dropbox 出现了一个 bug，任何人都能用任何密码访问任何账户，这一事故持续了 4 个小时 [参见 InformationWeek 的文章（http://www.darkreading.com/vulnerabilities-and-threats/dropbox-files-left-unprotected-open-to-all/d/d-id/1098442）]。当然，所有供应商都会告诉你你的应用数据现在是安全的，他们已经提高了安全性。如果你认为这没什么关系，那么黑客还将会在云存储服务里找到漏洞，或者服务本身会再次出错。

> 大部分云存储的主要问题在于其天生是公开的。例如，商业版的 Dropbox 听起来非常好，并且确实提供了更多的安全特性，但该服务仍然是公开的。公司不能在自己的私有云上使用该服务。
>
> 此外，大多数云服务宣称它们会在自己的服务里加密数据，这似乎是真的。但是，这些服务提供商通常以必须用相关的凭据进行验证后才能访问你的数据为借口，持有着加密密钥。因为你不拥有自己的加密数据的密钥，所以就不能控制对数据的访问，并且加密的作用也会比你想象的小得多。

Web 应用程序的安全是一件大事，因为大部分应用（即使不是全部）始终要有 Web 应用

程序的基础。用户想要他们的应用程序无处不在，而浏览器几乎是唯一能以高效的方式在众多平台上提供这种功能的方式。简言之，你必须从一开始就考虑云存储的问题。对于如何将云存储作为 Web 应用程序策略的一部分，有下面几个方案可供选择。

- *阻止访问*
  通过使用防火墙，指定策略或应用程序的特性，确实可能阻止所有对云存储的访问。但是，想要阻止无处不在的想要访问云存储的用户去访问是很难的，并且，用户访问的意愿是相当坚决的。此外，阻止访问会对业务需求产生负面作用。例如，合作伙伴可能会选择云存储来作为交换大文件的方法。阻止策略还会招致用户的愤怒，进而使他们不再使用你的应用程序，或者想办法规避你提供的这个特性。当你的公司必须管理大量敏感数据，有保护数据的法律要求，或者根本不需要用到云存储的灵活性时，阻止访问是最好的选择。

- *允许不受控的访问*
  你可以选择无视云存储使用过程中涉及的相关问题。但是，这样的策略会让公司面临数据丢失、数据泄露以及各种各样的其他问题。不幸的是，许多公司目前正是使用这样的方法，因为控制用户的访问已经变得非常困难，且这些公司缺乏采用其他方法的手段。

- *依赖公司授权的安全场景*
  如果要求用户用公司账户访问云存储，你至少能监控文件的使用并能在职员离职时恢复数据。但是，云存储的基本问题仍然存在。有相关知识的黑客仍然能侵入账户并窃走数据，或者只是以其他方式来窥探你。在你的公司不需要管理法律要求保护的数据，而你也愿意用一些安全性换取方便的时候，这一方案是可行的。

- *控制应用程序的访问*
  许多云服务支持以特别的方式与服务进行交互的 API。虽然这一方法非常耗时，但它确实能让你控制将用户敏感数据保存在哪里，同时用户仍然能够灵活地使用云存储处理不敏感的数据。在公司需要与大量合作伙伴进行交互，还需要管理大量敏感和核心数据时，你应该考虑这一方案。

- *依赖第三方的解决方案*
  你能找到一些第三方解决方案，比如 Accellion（http://www.accellion.com/），它提供了云存储的连接器。这种供应商提供的服务可以作为应用程序与在线数据存储之间的中介。用户可以无缝地与数据进行交互，但连接服务会根据你设置的策略来控制访问。这种方法的问题在于，在开发应用程序时你有额外的一层需要考虑。此外，你必须信任提供连接服务的第三方。在需要拥有灵活性而又不消耗一般的开发成本，并且不想创建依赖于 API 访问的解决方案时，你可以使用这种解决方案。

1.6 使用外部代码和资源

当今的大多数公司都不具有完全从头开始搭建一个应用程序所需的时间与资源。此外，维护这样一个应用程序的成本会很高。为了让成本处于可控范围，公司通常依赖各种形式的第三方代码。这些代码会执行基础任务，而开发人员会像玩乐高玩具一样使用它们组成自己的应用程序。但是第三方代码并不会消除安全方面的挑战。事实上，你正在依赖别人为

你的应用程序写出不仅运行良好、能执行所需的所有任务，而且还很安全的代码。接下来会探讨在使用外部代码及资源时会遇到的一些问题。

1.6.1 定义库的使用

库是任何你添加进自己的应用程序的代码。许多人将库定义得更加广泛一些，但对本书而言，其要点是，库包含着代码，并且在应用投入使用时其已成为应用程序的一部分。一个常用的库是 jQuery（https://jquery.com/）。它为应用程序提供了丰富的基础功能。有趣的一点是，在 jQuery 里术语库和 API 是可互换使用的，如图 1-2 所示。

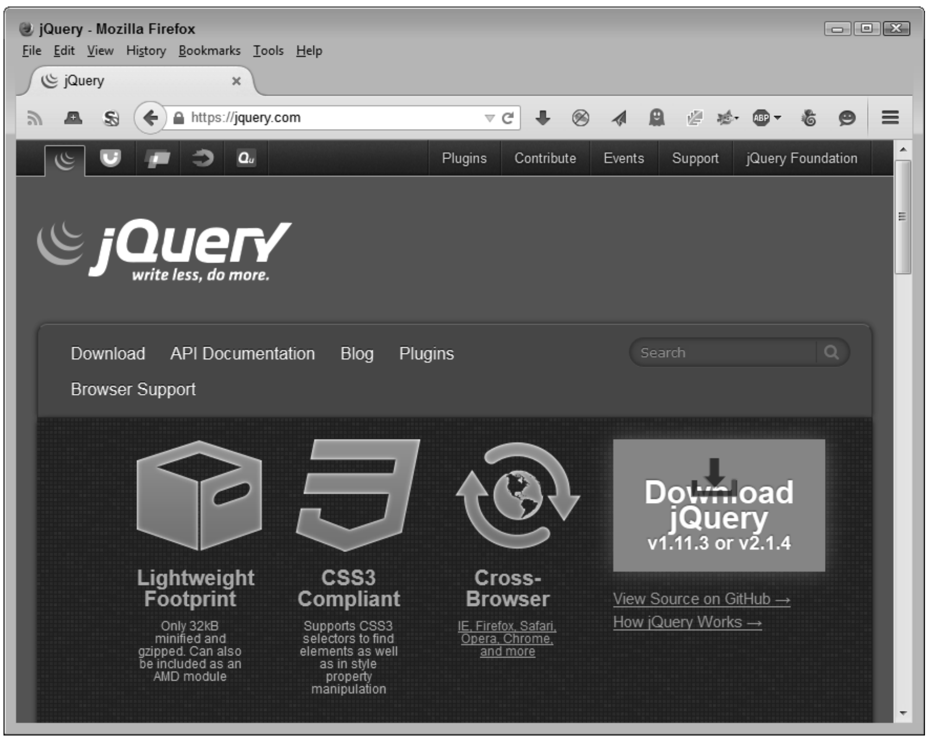


图 1-2：很多网站把库和 API 互换使用

jQuery 的网站还会告诉你关于这个代码库的最佳配置。事实上，jQuery 展示其本身的方式为你想使用的任何库做了个很好的榜样。它有全面的文档，并且你能找到每个调用的多个例子（以确保你至少能找到接近于你想使用的场景的例子）。更重要的是，这些例子是实时在线的，所以你能真实地看到这些代码在与应用程序计划采用的同一款浏览器上的效果。

> 像任何其他软件一样，jQuery 也有自己的缺陷。随着进一步阅读本书，你会了解其他的库以及关于它们的更多细节，从而开始发现功能与安全是如何一起运作的。因为 jQuery 是一个如此大而复杂的库，所以它提供了很多功能，但也提供了更多黑客可以利用的攻击面。

当使用库时，安全工作的重点是你的应用程序，因为你从主机服务器上下载用于应用程序的代码。库的代码是在进程内执行的，所以你需要明确获取库代码的源头是可信任的。第 6 章探讨了使用外部库作为应用程序开发策略一部分的复杂性。

1.6.2 定义API的使用

API 是任何可以像进程外服务一样调用的代码。如果你发送一个请求给 API，那 API 就会响应你所请求的资源。资源通常是某种类型的数据，但 API 也执行其他类型的任务。其特点是，代码驻留在其他系统里而不会成为应用程序的一部分。因为 API 是以请求 / 响应方式工作的，所以它们能提供比库更广泛的平台支持，但其运行速度会比库慢，因为其代码与使用它的系统并不在一个地方。

API 的一个优秀例子是 Amazon 提供给不同开发人员的服务（https://developer.amazon.com/）。图 1-3 展示了其中的少量服务。你必须为你想用的每个 API 进行注册，且大多数情况下 Amazon 会给你一个特殊的密钥。因为你正与 Amazon 的服务器进行交互，而不是简单地把代码下载到自己的系统中，所以使用 API 时应遵从的安全规则是不一样的。

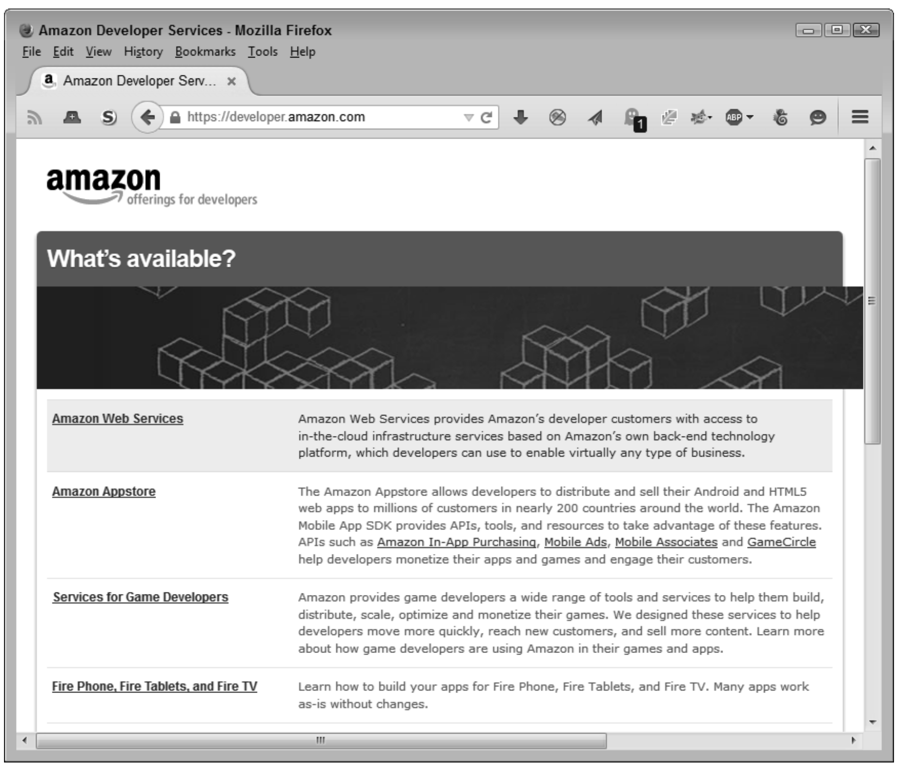


图 1-3：Amazon API 是一个代码在外部主机服务器上执行的例子

每个 API 都有自己的生命周期且依靠不同的方法去处理管理数据等问题。因此，当比较两

个 API 时，即使它们都来自同一家机构，你也不能对其中一个作出任何安全性方面的假设。

API 还依赖信息交换。任何信息交换都要求有额外的安全保障，因为你的部分数据始终会落在自己的主机系统上。你需要明白主机应提供适当的安全措施来保护请求中传递的数据。第 7 章探讨了如何安全地将 API 作为应用程序开发策略的一部分。

1.6.3　定义微服务的使用

与 API 类似，微服务也是在主机系统上运行。你发出请求，微服务则响应某种类型的资源（通常是数据）。但是，微服务与 API 差异巨大。微服务的重点放在小型任务上，一个典型的微服务专注于把单一任务执行好。此外，微服务往往很关注平台的独立性，其特点是，提供的服务要能与任何规模、具有任何能力的任何平台进行交互。API 与微服务关注点的不同会极大地影响安全形势。例如，API 往往更具安全意识，因为其所在主机能对发起请求的平台作出更多假设，并且一旦出问题，损失也会更大。

目前的微服务也往往是自产自销的，因为这一技术还比较新。看看现在提供 API 的网站，一旦该技术发展起来，它们也会开始提供微服务。与此同时，看看诸如微服务架构（http://microservices.io/）这样的网站，仔细审查微服务有何不同之处是值得的。这个网站提供了示例应用以及关于当前在线代码不同使用模式的讨论，如图 1-4 所示。

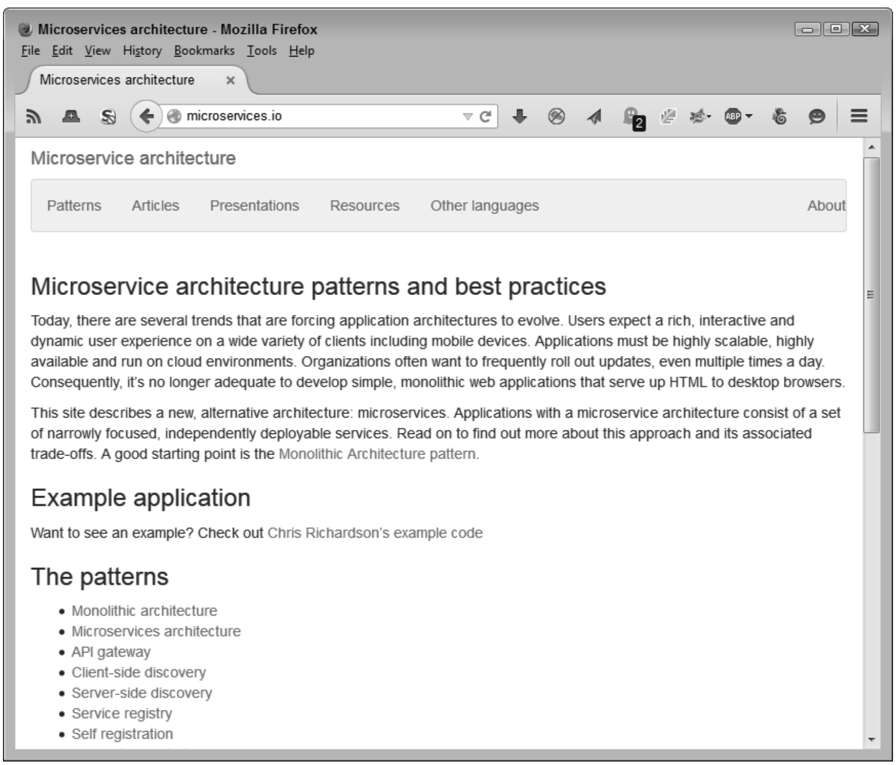


图 1-4：微服务是一门新技术，而该网站会通过不同的使用模式来帮你认识它们

当使用微服务时，你需要确保其所在的主机是可信赖的，且微服务执行的单个任务是明确定义的。还有一点很重要，要考虑微服务如何与你提供的任何数据进行交互，且不要假设每一个微服务都以相同的方式与数据进行交互（就算这些微服务存在于同一台计算机上）。微服务的使用意味着效率以及在更广泛平台上工作的能力，但你还需要意识到额外的安全检查的要求。第 8 章探讨了如何安全地将微服务作为应用程序开发策略的一部分。

1.6.4 访问外部数据

外部数据有各种各样的形式。任何形式的外部数据都是可疑的，因为有人会在单纯的数据访问行为中对数据进行篡改。但是，对数据按其可见级别进行分类，对我们考虑安全要求的相关事宜是有帮助的。

你通常可以认为私有数据源是相对安全的，不过你仍然需要检查数据是否存在潜在的有害元素（比如数据库字段中的加密脚本）。但是大部分情况下，数据源本身并不会刻意尝试引起问题。下面是最常见的作为外部私有数据源的信息存储来源：

- 存储在你自己公司主机上的数据
- 存储在合作伙伴计算机上的数据
- 由在服务器上运行的应用程序计算出的数据
- 从传感器或公司提供的其他数据源引入的数据

付费的数据来源也是相对安全的。任何提供付费数据访问服务的人都希望维持与你的关系，并且声誉在这一领域是最重要的。就本地和私有数据而言，你需要验证数据不会损坏或存在潜在威胁，比如一些脚本。但是，因为数据还需要在公共网络上传输，所以你需要检查是否存在由诸如中间人攻击这样的原因而引起的数据伪造和其他潜在问题。

> 你可以找到很多有趣的在线数据仓库，当你开发应用时，它们很有用。不用自己创建数据或依靠付费渠道，你通常可以找到一份免费的数据源。Registry of Research Data Repositories 这样的网站现在就提供 API，让你能更准确地搜索到适合的数据仓库。在此情况下，你能在网页 http://service.re3data.org/api/doc 上找到相关的 API 文档。

数据仓库可能会存在问题，并且仓库越公开，就越会产生问题。数据仓库的使用方式确实有所区别，但确保你得到的是真正的而不是伪造的数据，这是最重要的。大多数情况下，你能在使用数据之前将其下载下来并进行扫描。比如，图 1-5 所示的世界卫生组织网站就提供了筛选数据仓库的方法，让你能够准确地找到所需的数据，然后你可以仅下载那部分数据集，从而降低得到不想要的数据的风险。

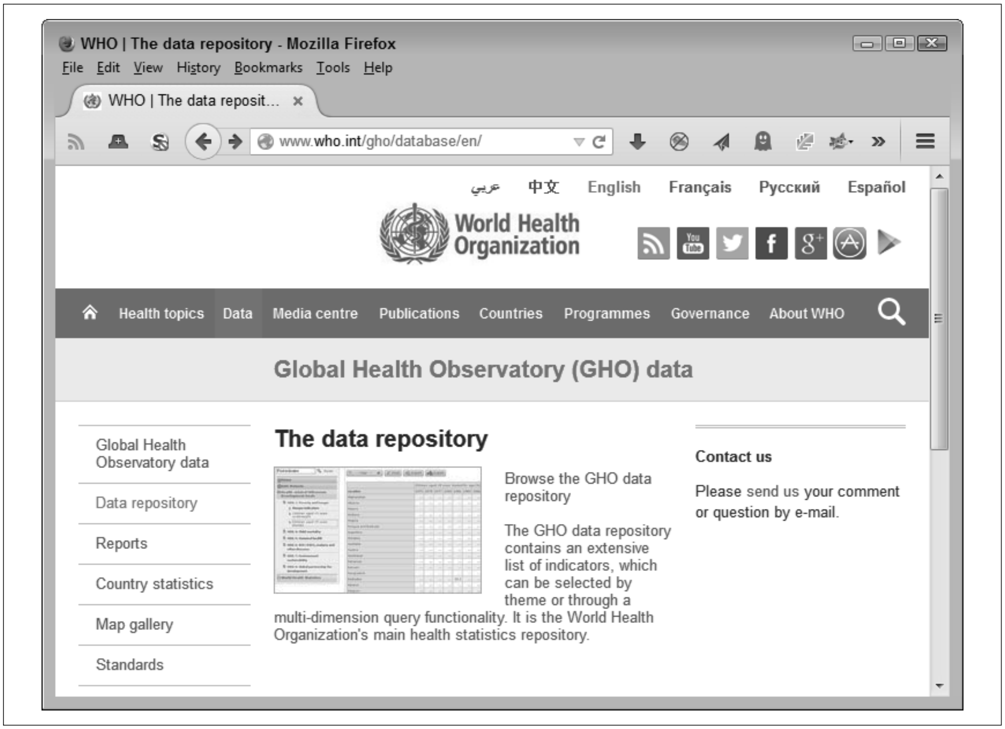


图 1-5：有些数据仓库（比如世界卫生组织的数据库）是可下载的

很多数据仓库和方法可以访问它们所存储的数据。你所使用的技术依赖数据仓库的接口以及应用程序的需求。请务必阅读第 3 章中与数据仓库有关的内容。第 6 章、第 7 章和第 8 章讨论了如何在库、API 和微服务中使用外部数据。每一个环境都有不同的要求，所以准确地理解代码会如何影响访问数据的方法，进而需要什么样的安全措施来保障无害地使用数据，这是很重要的。

1.7　允许他人访问

本章绝大部分内容讨论了对资源的保护，或者对由他人提供的数据和资源的使用。企业不是存在于真空中。当你创建了一份数据源、库、API、微服务或别人能用的其他资源时，第三方通常就会请求访问它们。只要允许第三方访问这些资源，你就将自己的网络暴露给了你可能永远无法想象的潜在安全威胁。商业伙伴也许是其他实体，它们似乎相当可靠且不会带来问题。但是，它们的病毒会变成你的病毒。它们面临的任何安全威胁都会变成你要面临的安全威胁。如果有很多第三方在使用你的资源，那么很可能至少有一个存在安全问题并且会导致你也出问题。

当然，在做任何事之前，你需要确保提供给外部的东西与你想象的一样好。确保提供的应用程序的安全性是很重要的。作为资源的提供者，你突然成为多个外部实体的单一故障点。维持环境安全意味着要完成以下事项：

- 定期检查所有资源以确保它们是可用的
- 提供有用的资源文档
- 确保第三方开发者遵守规则（通过在采购语言里加入安全性要求等）
- 执行必要的安全测试
- 持续跟进资源存在的潜在威胁
- 更新主机资源以避免已知的漏洞

将资源提供给外部访问的开发者还需要考虑其他的问题，但这些问题对于任何外部访问的场景来说都是很普遍的。此外，你必须期望第三方去测试资源以确保他们是按所给建议操作的。例如，当提供库、API 或微服务时，你必须期望第三方会执行输入和输出测试，而不是简单地相信你所说的，资源会按预期运行。

一旦通过了将资源提供给第三方使用的初始阶段，你还必须维护这些资源，以便应用程序能继续依靠其运行。此外，想象你在提供资源时面临着某些威胁是很重要的。下面是一些你必须考虑的事项：

- 黑客会尝试利用你的资源来入侵你的网站
- 开发人员会滥用资源，并尝试用它执行其设计目的以外的任务
- 资源会以某种方式受损

第 2 章

迎合用户需求与期望

除非你可以说服用户接受安全，否则它将无法奏效。在得不到用户支持的情况下，任何苛刻的设备开发者想设法增强安全性最终都会失败，因为用户善于寻找绕过安全的方式。当安全性足够完备，可以阻挡除顽强用户之外的所有用户的尝试时，用户会拒绝使用该应用程序。大量失败应用程序的案例证明了一个事实，即增强安全性时一定需要某种程度的用户协助，所以，最好让用户参与你作出的任何决定。

用户对应用程序有两个级别的需求，且安全性必须满足这两个级别的需求。用户需要拥有足够的自由来完成工作所需的任务。当一个应用程序无法满足用户的需求时，它就是一个失败的应用程序。用户的期望则是额外的需求。用户希望应用程序能在公司设备之外的个人设备上使用。根据应用程序的不同情况，确保应用程序及其安全性能在最广泛的平台上工作，能为你带来良好的声誉，从而更容易让用户接受安全性方面的要求。

本章探讨了需求和期望，因为它们与安全性相关。对许多开发者来说，编写代码的目的是创建运行良好且满足所有需求的应用程序。但是，编写代码的真正目的是要创建出能让用户成功且安全地与数据交互的环境。虽然数据是应用程序的焦点，但用户才是触发面向数据的修改的手段。

2.1　从用户的视角看待应用程序

用户和开发者常常就安全问题发生争执，因为他们看待应用程序的角度完全不同。开发者关注的是实现各种有趣功能的精雕细琢的代码，而用户关注的是达到最终目的的手段。事实上，用户可能根本不关注应用程序。用户关心的只是在截止日期前得到一份报告或其他产品。对于用户来说，最好的应用程序是不可见的。当安全性成为阻碍应用程序变得不可见的因素时，它就会变成用户想要规避的问题。总之，让应用程序及其相关的安全性尽可能地对用户不可见是最好的，离这一目标越近，用户就会越喜欢使用你的应用程序。

当前事件确实会让开发者和用户产生争执。比如，Ashley Madison 网站被黑（http://www.computerworld.com/article/2982959/cybercrime-hacking/ashley-madison-coding-blunder-made-11m-passwords-easy-to-crack.html）就让开发者特别担心，并不太可能对这样的事妥协。在某些情况下，UI 和用户体验专家可以在开发者和用户之间调解，提出一个对双方都有利的方案。当涉及安全问题时，若双方情绪激动且某些解决方案对其中一方来说不可接受时，打破常规进行思考是很有价值的。

开发人员的问题在于他们真的不会像用户那样思考。抽象技术的炫酷特性让开发人员无法抵制。一个开发团队应至少包含一位用户代表（组织中一位真正能代表典型用户的人）。此外，你应该将用户作为测试流程中的一部分。就像应用程序里的其他 bug 一样，如果在开发阶段的早期就发现不起作用的安全措施，修复起来就更容易。如果一项安全措施对用户来说是麻烦的、繁重的、显而易见的或仅仅是令人厌烦的，那它就是失败的，即使其确实能保护用户数据。

虽然本书没有讨论开发运营（development and operations，DevOps）的开发方法，但你应该考虑将其作为应用程序设计与开发策略的一部分。开发运营（开发和运营的组合）强调沟通、协作、一体化、自动化，以及任何应用程序开发过程中的利益相关者之间的合作。你可以在网站 http://devops.com/ 上找到很多关于开发运营的资料。很多人尝试描述开发运营，但描述比较清晰的是 Patrick Debois 的一篇博文（http://www.jedi.be/blog/2010/02/12/what-is-this-devops-thing-anyway/）。虽然这篇文章有点老旧，但它很好地概述了什么是开发运营，它解决什么问题，以及你该如何在自己的应用程序里使用它。

安全问题确实是一家公司必须解决的问题。如果作为开发人员，你是唯一一位尝试创建安全解决方案的人，那该方案几乎肯定要失败。应用程序的用户视角对于让用户参与定义应用程序的安全策略是很关键的。但是，创建解决方案还必须让以下人员参与：

- 管理层（确保公司的目标达成）
- 法务人员（确保数据保护措施满足政府要求）
- 人力资源（确保你不会伤害任何人的感情）
- 技术支持（确保进行了所有需要的培训）
- 所有参与定义控制数据管理的经营策略的利益相关者

毕竟，执行数据管理的规则就是安全的全部意义所在。安全不只是确保黑客不会发现保存数据的隐藏文件夹。安全是对数据的可靠保护，这样用户就能对其做出负责任的改动，破坏性的改动则被规避了。

2.2 考虑自带设备的问题

用户会从家里带来自己的设备并用其访问你的应用程序，要习惯这一情况。理论上来说，你能创建检测哪些设备正在访问你的应用程序的方法，但事实上，在许多情况下用户会找

到绕开这些检测的方法。与在有明确硬件配置信息的条件下进行开发相比，开发能在自带设备（Bring Your Own Device，BYOD）环境中运行良好的应用程序要困难得多，但可以达到更好的效果。关键是要假定用户会依赖他们自己的设备，并以此为目标实现应用程序安全性。接下来描述了在处理 BYOD 时会面临的问题以及解决这些问题的一些潜在方案。

> 为了省钱，有些公司确实接受了 BYOD，这意味着这些公司里的开发人员没有用于测试的标准设备。这类公司简单地认为用户拥有同时适用于工作和娱乐的设备。如果你的公司是其中一员，那你不仅需要应对把 BYOD 设备作为公司设备替代品的情况，并且还需应对将 BYOD 设备作为唯一选择的情况。了解用户拥有哪些类型的设备是值得的，这样对于将哪些种类的设备用于应用程序的测试以及如何设置使用应用程序的设备标准，就有了决策的基础。

2.2.1　理解基于Web的应用程序的安全性

最有可能实现 BYOD 的方案就是为每一个需求创建基于 Web 的应用程序。然后用户就能依靠智能手机、平板电脑、笔记本电脑、PC 或任何安装了合适浏览器的其他设备访问应用程序。但是，基于 Web 的应用程序也很难保障安全。想想那些为所有设备提供除密码安全外的所有功能的需求。事实上，密码这个词在这里的真正意义甚至可能不是一个密码，在你的情况里可能只限于指个人身份号码（personal identification number，PIN）。对于基于 Web 的应用程序来说，大多数情况下最薄弱的安全环节是移动设备。为了让移动设备更安全，你需要考虑执行以下这些步骤。

(1) 让应用程序的所有利益相关者（包括用户、CIO、CISO、人力资源以及开发团队以外的其他人）参与到应用程序特性的决策过程中来。你应该将此作为第一步，因为这里的每个人都会帮你制定聚焦于用户和业务需求的策略，还能让你找出与未满足的期望（不是需求）相关的问题。

(2) 以书面形式制定一个移动安全策略。如果在会议期间商定协议但没有以书面形式记录下来，人们往往会忘记商定过什么，并且这会在后面的流程中变成问题。一旦你有了一个书面的策略，就要确保每个人都知道它并已阅读过它。这对于正在进行应用程序设计的开发人员来说特别重要。

(3) 确保管理层理解为安全措施提供资金支持的需求。如今的大多数公司在面对安全问题时都存在资源缺乏的情况。如果一个开发团队缺少相关资源去构建安全的应用程序，那么这些应用程序就会有很多可供黑客利用的漏洞。对开发工作的资金支持必须在开发进程开始前就到位。

(4) 获取正确的工具来创建安全的应用程序。开发团队从一开始就需要有合适的工具，否则将不能取得想要的结果。在大多数情况下，开发人员对安全目标会力不从心，因为他们缺乏合适的工具来实现安全性策略。以下是一些通常会用到的与解决该领域问题相关的工具：

   a. 用户或系统认证
   b. 数据加密
   c. 移动设备管理
   d. 通用的反病毒保护
   e. 虚拟私有网络（virtual private network，VPN）支持（需要时）

f. 防止数据丢失
g. 主机入侵防御

(5) 与有强大安全专业技能的公司建立合作关系（如果有必要）。在大多数情况下，公司会缺乏有适当专业技能的开发人员。从已经成功部署大量基于 Web 的应用程序的公司那里获取这些技能，会让你的公司节省时间与工作量。
(6) 开始开发工作。只有在为应用程序创建了强大的支持系统之后，你才能开始开发工作。当你遵循这些步骤时，就能够创建出一个环境，使得安全性从一开始就是 Web 应用程序的一部分，而不是在后期加上去的。

> 一份基于 IT 和安全专业人员的 2014 年 IDG 研究服务报告描述了大量与移动设备使用有关的问题。最热门的问题（75% 的专业人士投票）是数据泄露，这是公司交给开发人员通过应用程序约束来防止发生的事情。设备的丢失与被盗有 71% 的人投票，紧接着是不安全的网络访问（56%）、恶意软件（53%）以及不安全的 WiFi（41%）。

2.2.2 考虑原生应用的问题

对基于 Web 的应用程序相关的问题的直接反应就是用原生应用程序替代。毕竟，开发人员很了解该技术，且有可能使用操作系统的安全特性来确保应用程序能如最初预期般保护数据。但是，原生应用程序的时代已进入倒计时。随着代码变得更加复杂，支持原生应用程序变得更加困难。此外，提供从众多平台访问应用程序意味着你能获得以下的重要益处：

- 提升工作人员之间的协作性
- 增强客户服务
- 在任意地点访问企业信息
- 提升工作效率

当然，支持众多平台有许多其他好处，但是这一清单指出的是使用原生应用程序的问题。如果你真的想用原生应用程序来确保更好的安全性，就需要为每一个想支持的平台开发一个原生应用程序，这会变成一个巨大的工程。对大多数公司来说，从增强安全性和提升应用程序控制的角度出发，花费时间开发所需的这些应用程序是不值得的。

> 有些公司尝试用混合方式绕过原生 / 基于 Web 的应用程序这一问题，但这一方式在许多情况下工作得并不好。例如，给原生应用程序使用一个 iOS、Android 或是基于 Web 的接口很容易出错，并会带来潜在的安全问题。使用纯基于 Web 的应用程序当然会更好。

2.2.3 使用定制化浏览器

在开发基于 Web 的应用程序时，你可以走定制化浏览器的道路。在某些情况下，这意味着写一个包含浏览器功能的原生应用程序。原生应用程序会提供额外的特性，因为它可以将安全做得更好，同时基于 Web 的应用程序还能让手机访问不太敏感的功能，这会让用户非

常开心。有些语言（如 C#）提供了立即可用的、相对实用的定制浏览器能力。但是，使用任何一种应用程序语言都能创建定制浏览器。

> 和组织讨论将在智能手机和平板设备上使用锁死开关（kill switch）作为应用程序开发策略的一部分，这是个很好的主意。锁死开关能让被盗的智能手机成为无用的垃圾。根据《今日美国》上的一篇文章（http://www.usatoday.com/story/tech/2015/02/11/kill-switch-theft-down/23237959/），锁死开关的使用极大地降低了几个大城市的手机偷盗率。*PC World* 上最近的一篇文章（http://www.pcworld.com/article/2367480/10-things-to-know-about-the-smartphone-kill-switch.html）提供了一些关于帮助管理层了解锁死开关是如何工作的相关信息。在许多情况下，你必须安装相关软件以使锁死开关可用。使用锁死开关可能听起来很极端，但比起让黑客访问敏感的企业数据要好一些。

除了直接的安全项，定制化浏览器的解决方案还让间接的安全选项更简单。虽然在应用程序中创建的定制浏览器可提供完整的功能，但应用程序开发者可选择不实现某些特性。例如，你可以选择不允许用户输入 URL 或依靠历史功能在不同页面间切换。应用程序仍然会像任何其他浏览器一样展现页面，但不能以标准浏览器的方式控制会话。这层控制使得访问更敏感的信息成为可能，因为用户将不能进行通常会导致病毒下载、信息泄露或破坏应用程序环境的那些事情。当然，每一条规则都会有例外。如果一种针对浏览器库和函数以窃取信息的病毒已经存在于系统内，那该病毒很可能仍旧可以获取由使用了该库的定制浏览器管理的内容。

定制浏览器的环境还让开发者有机会使用在一般情况下不能工作的那些编程技术。在让第三方库、框架、API 和微服务正常运行方面，开发者不断遇到问题，因为不是每一个浏览器都能提供必要的支持。例如，为了检测某个浏览器是否支持你想使用的 HTML5 特性，你需要使用 HTML5test（https://html5test.com/）这样的网站来检查它，以获取潜在问题领域的列表，如图 2-1 所示。

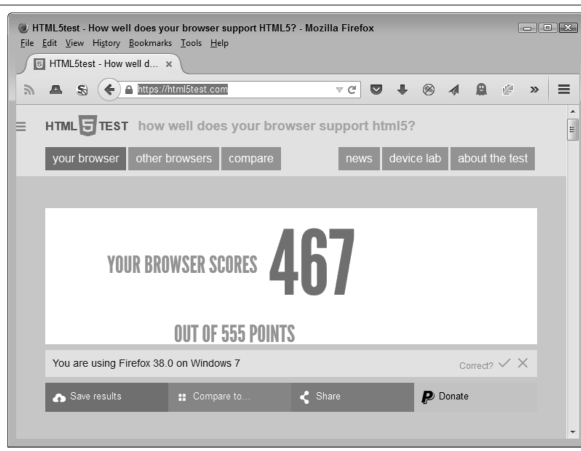


图 2-1：使用标准浏览器意味着寻找潜在支持问题

定制浏览器方案的问题在于，用户选用的设备不同会引起用户支持的差异。当这些差异存在时，开发者通常会遭受惩罚。用户会想在最公开的地点用最不安全的设备访问公司最敏感的数据。想象在星巴克用一部智能手机处理病人记录的情形。数据可能最终会出现在任何地方，而数据泄露将几乎肯定会发生。在某些情况下，开发人员需要与从管理层往下的每个人合作，以制定出一份合理的数据处理措施清单，这意味着使用定制化浏览器的解决方案将不会太流行，但它会有利于实施稳健的数据管理策略。

2.2.4　验证代码兼容性问题

BYOD 现象意味着用户会使用各种各样的设备。当然，这是一个问题。但是，更严重的一个问题是，用户会在这些设备上使用老旧的软件，因为旧版本的软件用起来比较舒服。使用老旧软件的后果是，应用可能会出现问题，但这些问题不在于应用程序，而是由非常陈旧的软件导致的代码兼容性问题。为此，你需要依赖 HTML5test（在前一节介绍过）这样的解决方案来检查用户的软件，以确保其满足最低要求。

解决该问题的另一个方法是，在开发应用程序时发现潜在的代码兼容性问题。例如，图 2-2 展示了提供 HTML5 文档的网站 W3Schools.com（http://www.w3schools.com/tags/）。网页的底部有一个支持某一特性的浏览器列表以及支持该特性所需的浏览器版本号。通过在开发应用程序时跟踪这些信息，你有可能减少代码兼容性问题。最起码，你能够告诉用户必须使用哪些版本的浏览器才能在自己的设备上运行你的应用程序。

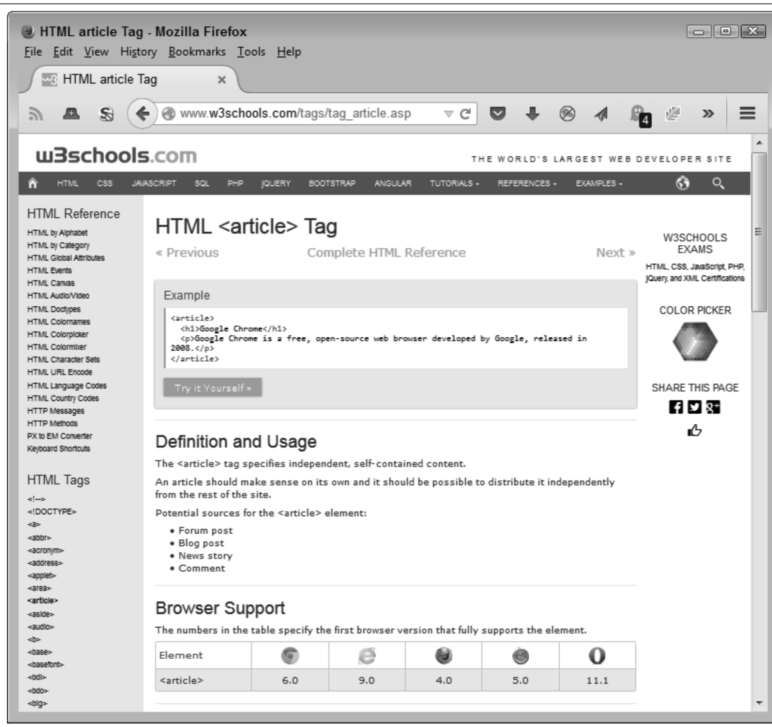


图 2-2：验证你计划使用的代码特性在用户浏览器上是否能正常运行

另外一件很重要的事情是，要注意有些网站会以简明的方式告诉你一些兼容性问题。W3Schools.com 这个网站也提供了这一功能。注意，图 2-3 展示的 HTML 标签清单会告诉你哪些是 HTML5 支持的标签，哪些不是。掌握了这些信息，就能在编码阶段节省大量时间，因为你不必浪费时间尝试找出为什么某一特性在某个用户系统里不能按预期方式工作。

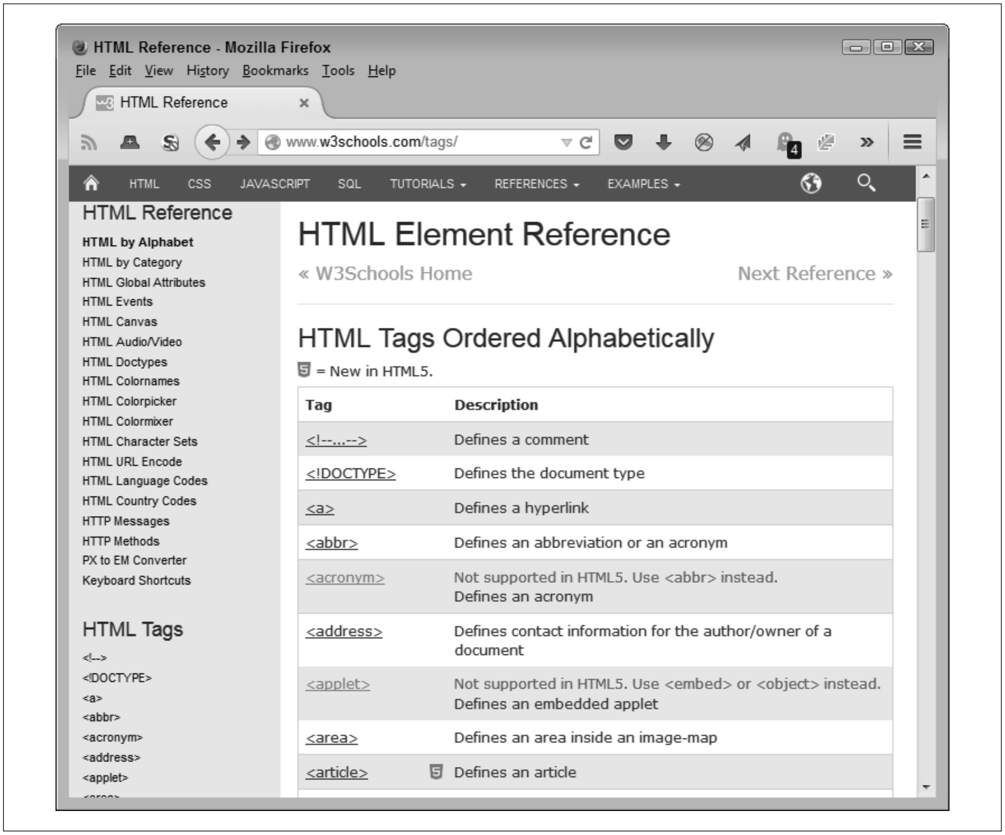


图 2-3：找出能以简明方式告诉你版本问题的文档很重要

大多数开发人员会体验到的一个严重的代码兼容性问题是，让 HTML5 在老旧的浏览器上正常显示。比如 http://www.w3schools.com/html/html5_browsers.asp 这样的网站为你提供了一些可用的答案。例如，它展示了如何用 `html5shiv` 来让 IE 浏览器支持 HTML5 元素。cdnjs（https://cdnjs.com/）网站包含各种各样这类有用的 JavaScript 插件。你能够在网页 https://cdnjs.com/libraries 上找到它们。图 2-4 只展示了可用库的一小部分清单。不幸的是，你需要自己为所有这些库找出例子，因为该网站没有提供太多有用的信息。多数的代码库文档在 GitHub 上。例如，你可以在网页 https://github.com/afarkas/html5shiv 上找到 html5shiv.js 的文档。

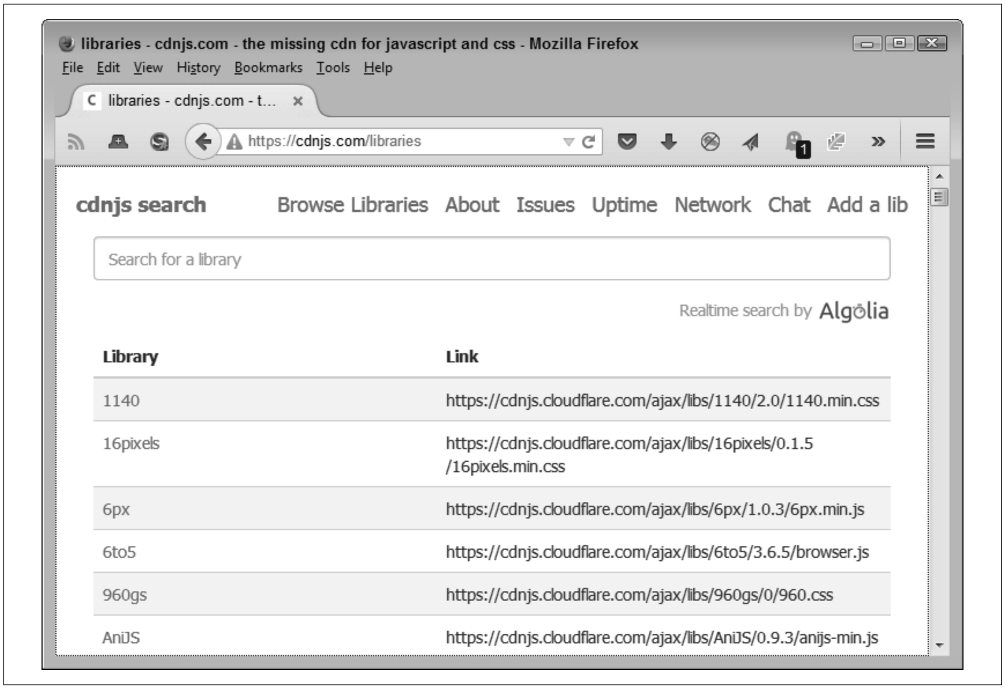


图 2-4：cdnjs 网站提供了大量有用的库

> 你会发现 CDN 这一缩写在网络上随处可见。CDN（content delivery network，内容分发网络）是提供各种类型网络内容的一系列服务。CDN 的目标是提供高度可用性及快速的分发。当需要的时候，它还可以提供区域适配的内容。所以，cdnjs 就是一个专门设计用来获取 JavaScript 代码并让它可为广大开发人员所用的 CDN，就像 Google CDN（https://developers.google.com/speed/libraries/）所做的那样。

2.2.5　处理几乎连续的设备更新

你的应用程序需要足够灵活以处理各种奇怪的场景。如今比较常见的一个场景是处理几乎连续的设备更新。在这种情况下，用户可能前一天还非常愉快地在智能手机上使用你的应用程序，但第二天却不能了。常见的做法是责备用户，但大多数情况下用户是无辜的。整个问题的根源在于更新常常在用户没有同意或知情的情况下就发生了。不幸的是，一次更新可能导致以下问题：

- 代码兼容性问题
- 安全漏洞
- 设置丢失
- 设备无法启动
- 数据损坏

解决这一问题的一种方法是将关闭自动更新作为应用程序安全策略的一部分。在已经将更新作为推出过程的一部分进行测试之后才手动进行更新，是确保应用程序将继续运行的最佳方式。不幸的是，这一解决方案并不能奏效。除非发生了某些错误，否则用户不会拿他们的设备进行测试。就算用户愿意拿出自己的设备，你也可能不具有进行测试所需的资源。

另一种解决方案是将应用程序设计成在启动阶段自动对更新进行检查。如果应用程序运行所需的产品版本号发生了改变，应用程序能将这一信息发送给管理员，作为潜在解决方案的提醒。

创建灵活的设计也是处理不断更新问题的方法。虽然用高明的编程技巧来帮助保证数据安全看起来是一个不错的主意，但它可能并不是。坚持采用最佳开发实践的策略，在可能的时候使用标准库，将安全性保持在合理的级别，这些都有助于保证在用户设备发生隐蔽的更新后，应用程序还能够继续运行。否则，你可能会在宝贵的周末花费大量的时间尝试解决阻碍应用程序运行的安全性问题。

2.3 设计密码的可选方案

密码看起来是识别用户的直接方式。你很容易更改密码，让它们足够复杂以降低被别人猜出的可能性，且它们是非常轻量的。但是用户认为密码很难使用，甚至难以记住，并且会成为对你设置在某处的安全措施的痛苦提醒。（在这种情况下，那些必须不断重置密码的支持人员会同意的用户感受。）用户不想要输入密码来访问或使用应用程序的某些功能。遗憾的是，你仍然需要一些方法来确定用户的身份。接下来描述了一些成熟的方案（你如今可以实现的），这有助于你找到最合适的解决方案。

浅谈近场通信

技术上更有趣的发展之一是已为各种需求使用了近场通信（Near Field Communication，NFC）。NFC 是主要用于产品的无线射频识别（radio frequency identification，RFID）技术的扩展。你可以在所买的每一样商品上找到无源标签。RFID 电子标签还扮演着安全设备的角色，但其主要目的是识别产品。扫描器发出的无线电波会赋予电子标签能量，且电子标签会返回所包含的信息。NFC 是 RFID 的高频子集，前者提供了一些特别的特性，有助于你为应用程序创建安全解决方案，其中包括以下方面：

- 同一设备能同时作为发送者和电子标签
- 设备之间能安全地交换信息
- 能根据需要对设备再编程
- 设备包含智能与本地存储，这令其比 RFID 更灵活

NFC 提供了消除密码的可能。一个人可以有一个嵌入了芯片的信用卡大小的 ID 徽章。将此徽章轻触到支持 NFC 的设备上就能完成登录。今天已经有许多支持 NFC 的设备，其中包括 PC、平板电脑和智能手机，所以这一方案能够在各处生效。

因为这一技术目前还不成熟，所以仔细评估你的选择很重要。幸运的是，万维网联盟（World Wide Web Consortium，W3C）已经在制定 Web 开发的标准。你能在网页 http://www.w3.org/TR/nfc/ 上看到相关草案。符合这一标准的库、API 或微服务能提供必要的资源，以创建满足公司与用户需求的可靠应用程序，同时降低支持成本并让应用程序更易为用户所用。最妙的地方在于，NFC 的解决方案能让你创建真正安全的应用程序。

2.3.1 使用口令

密码难记。创建一个 `!jS2Zd5L8` 这样的密码会让黑客很难猜出，并且能提升应用程序及其数据安全的概率。但是，这样的密码也不太好记，所以用户经常将其写到某个地方。事实上，用户可能只是在硬盘上保存了一份文件以方便访问。当然，黑客知道这一情况并会查找这样的文件。总之，从安全的角度看，密码是一个很好的做法，但从用户的角度看，其是可怕的设计。但是，使用非常容易记忆的口令有可能可以实现相同级别的安全性。

口令是一种部分可读的语句。你可以组合字母、数字以及特殊字符，让密码更复杂。例如，你可以创建一个 `I luv fl0w3rs!` 这样的口令。这一口令包含了大写和小写字母、数字以及特殊字符。它能对抗字典和其他常用的暴力攻击。它可能会比多数用户采用的密码长。但是，其仍然很好记。用户不需要把口令写下来。

今天的用户确实有理由抱怨他们需要记住的密码的长度。一种可以考虑的作为解决方案的技术是密码保险箱。密码保险箱能以一种安全的方式保存大量用户密码，可让用户继续使用复杂的密码。

用户选择他们能记住的口令很重要，但不要与应用程序、用户的个人生活或其工作环境有关。我们没有理由让黑客在猜测密码时占据优势。所以，`I Work in R00m 23a.` 这样的口令确实不是很好，太容易被猜到了。

简单告知用户使用口令不会有什么效果。事实上，用户仍将使用 `secret` 和 `master` 这样的密码，因为他们已使用很长时间了。用户往往会拒绝让他们的工作变得哪怕一丁点儿不顺的事情。因此，你仍然必须在应用程序代码里执行复杂规则，以强迫用户在涉及密码时采用比一般的弱密码更好的密码。如果用户不得不使用一个复杂的密码，那么使用口令的做法就会变得更吸引人。

2.3.2 使用生物识别的方案

生物识别的解决方案依靠你作为人的某些唯一特征来生成密码。生物识别的观点是，生物学密码对于一个特定的人是唯一的，其不易被偷走，用户不会丢失它，且密码足以复杂到黑客无法猜测（至少不容易）。下面是三个最常使用的生物识别解决方案。

- 指纹
- 虹膜
- 声纹

这三种解决方案都有缺陷，黑客有攻克它们的方法，所以你可能要选择另一种生物识别方案或将其中一种与其他认证方式结合使用。相关供应商有一些正在开发中的生物识别替代方案，下面描述了其中的一些。

- 心跳

  一个更有趣的生物识别替代方案是结合带分析算法的心率监视器。事实上这种方案已经以 Nymi 智能手环（https://www.nymi.com/）的方式出现。该方案依靠 NFC 将用户的密码传输给任何支持 NFC 的设备。用同一个手环可以登录计算机，启用某个应用程序功能，打开房门，或者发动汽车。

- 人脸识别

  在电影《少数派报告》中，当人们走在街上时，摄像头扫描他们的脸并展示在广告里。有趣的是，这一技术已经以 Facebook 的 Deepface（https://research.facebook.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/）的形式出现。只要注视连接了摄像头的计算机，你就能登录系统并获得所有需要的应用程序功能。Facebook 最近称（http://money.cnn.com/2014/04/04/technology/innovation/facebook-facial-recognition/）其能从正面或侧面进行扫描，这使得该方案比其他生物识别方案更灵活。

  > 有趣的是，人们拍的所有自拍照会使执法机构及其他人非常容易地建立起一个人脸识别数据库，这能让任何人都无所遁形。只要想想每个商店里的摄像头都能根据人们的脸部特征将其识别出来所带来的影响。

- 耳形

  你像打电话一样把智能手机放在耳朵上。但是，你听不到别人对你讲话，而是登录进了一个应用程序。这一方案已经以 Ergo Lock Screen 应用程序（http://www.descartesbiometrics.com/ergo-app/）的形式出现。

- 输入识别技术

  每个人都有不同的输入方式。打字的速度、按住键的时间，甚至是输入字母的间隔，都能识别出作为打字者的你。通过输入一段特定的语句并监控你如何输入，应用程序可创建一种双因素认证，这与只是输入密码并没有什么不同。但是，现在黑客窃取到密码后仍然用不了它。一家已经实现该技术方案的公司是 Coursera，他们在 Signature Track（https://blog.coursera.org/post/40080531667/signaturetrack）中使用了这一技术。

生物特征识别技术的承诺并不能完全匹配现实中的生物特征。你从描述中会知道，黑客已经想出了攻克生物学密码的方法。如果用户的密码泄露，你只需要给他一个新的密码。但是，如果用户的指纹信息被泄露，实在不可能把他们的手指割下并接上新的手指。

为什么使用双因素认证

大多数单因素认证方案的问题在于它们有一个很容易被利用的弱点。例如，黑客可通过社会工程学的攻击或是暴力来窃取密码或口令。通过使用双因素认证，可以降低认证过程被攻克的风险。当然，有人会说三因素或四因素认证会更好，但那样会导致没有人能登录进自家账号的情况。

有很多类型的双因素认证。例如，你可以给用户一个密码和一个令牌。你还可以将密码与生物特征识别方案结合起来。许多银行目前就使用了双因素认证，并且你可以在 Google、Facebook 和 Twitter 这些网站上有选择地使用它。

双因素认证的问题与单因素认证的问题一样，那就是通常用户根本不喜欢身份认证。人们觉得应该可以在不做任何额外操作的情况下使用应用程序。当然，认证是重要的，并且你应该为核心或敏感的数据做双因素认证。但是，从用户的角度考虑选择最适合的认证方式，这也很重要。如果你想要安全方案得以成功，创建灵活的解决方案至关重要。

2.3.3 依靠钥匙卡

大部分人把钥匙卡看作一种较旧的技术，但它仍被广泛使用。例如，去住汽车旅馆时前台的人员可能会给你一个有特殊编码的磁条钥匙卡。钥匙卡替代了以前使用的钥匙。有些公司还会使用钥匙卡去满足不同的需求，例如控制访问如停车场这样的区域。一个钥匙卡与一个个人身份号码（PIN）的结合通常用来控制对敏感区域的访问。可以说你在生活中可能不止一次地使用过钥匙卡。

钥匙卡技术在持续地提升。一些现代的钥匙卡看起来根本不像一张卡，它们会以小饰物或其他设备的形式出现，用户能将其挂在钥匙链上或挂在脖子上。通过使用 RFID 或 NFC 技术，用户甚至不需要去刷某个设备，只需在锁前摇一摇就可以了。可是，钥匙卡的原理没有改变。你拥有一个包含了安全信息的物理设备，用户用它来访问资源而不是靠输入密码。

个人电脑也会使用钥匙卡技术。你能用它来控制对整个个人电脑的访问，或者控制对某个应用程序的访问。当用于访问某个应用程序时，开发人员需要开发读取卡信息的代码，检测其有效性，并认证用户。

使用钥匙卡的主要优点是，它能提供复杂的密码及潜在的其他细节来帮助识别用户。为了防范黑客，密码可以设得尽可能复杂。根据所使用的技术，你甚至可以很容易地改变钥匙卡的信息，所以密码的更改操作不会有太高的成本。钥匙卡通常也比较大，用户不会经常弄丢（虽然你可以预见有些用户至少会把一些钥匙卡忘在什么地方）。因为该技术由来已久，所以比起其他安全解决方案，钥匙卡的成本相对比较低。用户也比较倾向于选择钥匙卡（丢失时除外），因为它们比较快且易于使用。

钥匙卡的主要缺点是用户会忘记将其放在哪里或遗忘在家里。丢失的钥匙卡能为黑客提供

对公司造成真正损害的入侵机会。即使只是把钥匙卡遗忘在家里，提供一张临时卡也是种额外的支持开销。当然，还有在用户不再需要时回收临时钥匙卡的问题。

Secura Key（http://securakey.com/）是一个可查阅各类钥匙卡技术的网站。这个网站展示了很多你能使用的钥匙卡方案。你甚至能找到一些键盘，比如 IOGEAR 模式（http://www.iogear.com/product/GKBSR201/），它们有必需的钥匙卡读卡器作为键盘的一部分。其要点是减少使用应用程序所需的用户交互，以减少潜在的用户错误和数据泄露。

2.3.4 依靠USB key

USB key 实质上是一个包含一个或多个密码或令牌的闪存驱动器。你将 USB key 插入电脑并启动，电脑会使用其中的数据登录系统。同一个 key 中可以包含访问多个应用程序的多个密码。应用程序需要识别 key 中的密码，但该技术可以实现在不输入密码或任何其他信息的情况下登录应用程序。Google 正在使用 USB key 方案（https://www.technologyreview.com/s/510106/googles-alternative-to-the-password/），并且你会意识到其他厂家也会跟随这一做法。与钥匙卡相比，USB key 有以下显著的优点：

- 不需要获取新的 USB key 就能更改密码
- 能包含多个密码
- 总体成本低于使用钥匙卡
- 可以升级各类凭据

当然，USB key 有许多与钥匙卡类似的缺点。例如，用户不管是丢失了钥匙卡还是 USB key，结果都是相同的，该用户不再能登录应用程序而其他人可以。事实上，由于 USB key 比钥匙卡小，因此用户会更有可能丢失 USB key，并且泄露的不仅仅是一个密码。

2.3.5 实现令牌策略

你通常会用智能手机来实现一套令牌策略。某个网站向用户发送短信、图片或声音以作为登录计算机的方式。例如，Illiri（http://www.illiri.com/）网站会发送一段声音到用户的智能手机上，用户只要将其播放给计算机就能登录。同样，Clef（https://getclef.com/）网站使用图片做了同样的事情。在这两个案例中，你可以选择每次发送不同的令牌给用户进行登录操作，这意味着即使黑客窃取了某个令牌，也根本没有用。

在大部分情况下，在双因素认证系统中，应用程序会将令牌作为第二认证方式。第一认证方式仍然是钥匙卡、生物学特征、密码或口令。从理论上说，如果愿意，你也可以将令牌作为主认证方式。比起其他认证方法，这种方式会带来以下这些你需要解决的问题。

- 用户需要一直将智能手机带在身边。
- 丢失手机可能会损害作为登录应用程序方式的令牌系统。
- 该方案只能用于登录除智能手机外的设备，而如今的用户在可能的情况下都想使用智能手机而不是其他电脑设备。
- 用于用户登录的计算机需要有接收令牌所需的设备。

2.4 聚焦用户期望

在本章的这一节之前，你一直聚焦于用户需求。没有用户想要认证，但每个用户都需要认证。用户期望在开发中被归为“如果有，会更好”的类别，但实现它们会提升应用程序的友好度，这会让用户略微愿意使用你添加到应用程序中的必需项目。

当然，有些用户会有不切实际的期望，比如期望有一个令他们在别人面前特别有魅力的应用程序，或者一个能帮他们完成所有工作的应用程序，这样他们就能整天玩纸牌和看电影。遗憾的是，就算是世界上最优秀的开发人员也满足不了这样的期望。以下描述了一些合理的用户期望。

2.4.1 让应用程序易于使用

本章已经强调了易于使用的重要性。用户不想了解你所实现的安全方案的细节。事实上，用户根本不想了解关于应用程序的任何东西，除了它能提供用于完成与用户有关的任务所需的数据。从长远来看，你所做的任何让安全措施对最终用户不可见的行为，都会增大用户真正参与数据的安全管理的可能性，并且该策略会成功保证数据的安全。

与易于使用这一需求相对的是，创建一个如宣传所说的安全场景。只要有一处数据漏洞就能毁了公司的声誉并花费公司大量的费用。根据《计算机世界》杂志最近的一篇文章（http://www.computerworld.com/article/2926775/security0/data-breach-costs-now-average-154-per-record.html），当今的数据泄露平均每秒会造成 154.00 美元的损失。每条记录丢失造成的损失在持续增加，所以对安全解决方案的需求也在持续增加。用户的期望是易用，而应用程序的现实是需要在任何情况下保证数据的安全。

2.4.2 让应用程序快速运行

许多开发人员不理解用户不惜任何代价追求速度的需求。当然，部分问题在于过去几年保持用户注意力的时间在下降。根据《卫报》网站（http://www.theguardian.com/media-network/media-network-blog/2012/mar/19/attention-span-internet-consumer）的一篇文章，你有一到五秒的时间去获取用户的注意力。用户想要实时的响应和快速的修复。深入思考和深度分析已不再是用户字典里的一部分。不幸的是，你给应用程序添加的安全措施越多，通常它就会越慢。如果你问大多数用户在安全性方面想花多少时间，他们的答案会是 0。用户真的不关心安全性，他们只想更快地获取数据。

用户的期望是在获取执行某个任务所需的数据时不要有任何等待。并且，数据需要从一开始就是准确的且以用户需要的形式组织。否则，用户就会感到挫败。事实是安全措施会使得应用程序变慢，但你作为开发人员，需要专注于将延迟保持在最低的水平。

此外，一个安全的环境意味着不是所有数据都能一直被访问。你肯定不能允许一个用户坐在当地的星巴克里喝拿铁时查看病人的记录。用户的这种级别的访问是一种不合理的期望（并不是说要阻止用户提出这种要求）。所以，一个快速的应用程序应该能快速认证用户并提供合法的数据访问方式。隐藏非法选项常常能从一开始就防止用户发现它们，但你仍然需要提供一些帮助说明来解释为什么没有某些数据操作项。

2.4.3 创建可靠的环境

在考虑任何其他事项之前，你必须使应用程序可靠。应用程序不能以不可预测的方式运行，因为用户会遭受失败并不再使用它。就算应用程序有一点慢，但与那些运行很快但不能正确执行任务的应用程序相比，用户对速度的抱怨会更少。安全性与数据操作功能完美工作也是很关键的。第 5 章探讨了各种你能用来创建可靠应用程序的技术。本书的第三部分讨论的是各种类型的测试。你需要仔细进行测试以创建可靠的应用程序，这能减少用户的抱怨。

因此，用户期望是应用程序在任何情况下都能完美工作。再次强调，这需要让应用程序透明，让用户只需集中精力完成任务。在现实世界中，通过适当的编码技巧、测试技术、威胁分析以及对设备更新等问题的持续检查，你可以尽量接近完美。所以，在这一特定情况下，用户的期望能相当接近你在现实中能提供的东西。

2.4.4 客观看待安全性

如果你没有从本章中学到任何东西，那么重要的是要记住一个事实——安全是一个寻找平衡的行为。除非你能客观地看待安全性，否则你的应用程序可能不足以保护数据，或通过采用严厉但并没有用的措施来妨碍用户完成操作。用户不仅会很快发觉他们不喜欢的功能，还会发现不能达成任何有用目标的功能。

虽然用户的期望有可能是应用程序不需要有安全性，但大部分用户都能认识到安全是必需的，特别是如果用户想要让公司健康地运作。真正的问题不是让用户接受安全性需求，而是让他们接受特定级别的安全性。当客观地看待安全性时，你会很容易与用户达成协议并让他们站在你这边。

第3章

获取第三方帮助

重新造轮子通常是错误的做法。很有可能其他人已经创建了一个能很好地满足你的需求的安全解决方案，所以你能直接使用它而不需要创建自己的解决方案。第三方解决方案需要作为你的安全计划的一部分，这样别人才会知道你头脑中已有一个方案。此外，通过在安全计划中使用第三方解决方案，你可以包含各种相关信息并发起关于该方案的讨论。本章将探讨你该如何将各种第三方解决方案添加到自己的安全计划中。

当然，在做任何事之前，你必须找到你想使用的第三方解决方案。幸运的是，很多技巧可用于帮你减少这一过程所需的时间，让你研究真正能满足你的需求的解决方案。一旦找到你想使用的解决方案，考虑每一种方案类型的优点和缺点是很重要的。本章将第三方解决方案分为存在云端的方案和添加进应用程序的方案，比如库或 API。接下来的几节会让你明白，第三方解决方案确实能满足你的需求，并能减少创建一个有效方案所需的时间与工作量。

本章会讨论具体的例子，但所提供的信息适用于整个类别。例如，虽然 Capterra 在文中作为一个潜在的产品点评网站出现，但还存在其他类似的网站，你需要选择最适合你的应用设计及开发理念的网站。具体的例子只是用于帮助说明相关的原则。

3.1 发现第三方安全解决方案

找出一个第三方安全解决方案是很困难的。将能想到的大部分相关搜索词条输入搜索引擎，你能得到一份安全方案列表，其中的许多方案可能甚至与你的问题无关。搜索引擎给了你一个干草堆，而不是一根细针。不幸的是，磁铁还被拿走了，所以从中找到细针是不

可能的。你确实需要一种更好的方式以找到你想要的安全解决方案。

Capterra（http://www.capterra.com/network-security-software/）这样的点评网站会对你有所帮助，如图 3-1 所示。该网站提供了过滤功能，可以帮你搞清楚正在吸引你注意力的那些产品。每一个条目都有一段简短的点评，你可以点击进去找到更多的细节。在很多情况下，点评还会包含提供产品演示或其他有用信息的媒体资源。点评集中于一个地方，可用你指定的条件过滤出来，以基本上相同的方式格式化，这使得搜索过程变得更简单。



图 3-1：Capterra 这样的搜索网站让查找第三方资源变得更简单

验证你在 Capterra 及相似的点评网站上找到的信息是必要的。通过用网络搜索你感兴趣的产品以确定是否已有其他人点评过该产品，你能进行一定程度的验证。这些额外的点评所提供的见解有助于你作出更好的产品选择。

依靠杂志的点评也会有所帮助，这取决于你所选择的杂志。例如，*SC Magazine*（http://www.scmagazine.com/）经常点评对安全专家有帮助的产品。事实上，你可以将其中的一些点评发送到你的邮箱（http://www.scmagazine.com/events/section/109/）。该杂志甚至会赞助某种形式的竞赛（http://www.scmagazine.com/2014-sc-awards-us-finalists/section/3694/），以便你可以确定最可行的解决方案。

任何一种点评网站都会包含偏见。即使网站能公平地点评产品，采用你能检验的标准和你能验证的过程，一份点评仍然是某个人的观点，而你需要考虑作出点评的人是否掌握了足够的信息来为你提供你所需要的信息。经过一段时间之后，你会发现哪些网站在关于什么是真正的好产品上与你的观点一致。

在某些情况下，你能找到这样的组织，它们提供组织成员的快照，这些成员能为你提供一致的信息。例如，Cloud Security Alliance（CSA，https://cloudsecurityalliance.org/membership/solution-providers/）就属于这一类组织。通过查看这些网站，你能快速概略地了解参与到某个安全解决方案中的公司。当然，问题是你所读到的评论是由这些组织提供的，所以这些信息也是带有偏见的。可以肯定的是，支持这种组织的供应商已经排除了它们各种产品的负面评论。你需要谨慎地阅读这些信息并自己找出潜在的问题。

这种组织网站的好处在于你经常能发现其他有用的信息，因为让你参与其中是网站最大的利益所在。有些组织的网站还会告诉你将要发生的事件，你能从中获取关于潜在解决方案的额外信息，亲自与供应商见面，并发现你之前可能不知道的新技术。

有些在线杂志网站会推出一些特色文章，提供非常有用的信息。例如，《麻省理工科技评论》的文章“Improving the Security of Cloud Computing”（http://www.technologyreview.com/news/424298/improving-the-security-of-cloud-computing/）就讨论了可用来保持数据安全的技术。这篇文章提出的两个解决方案指向的是你可能没有考虑过的问题。在第一个方案中，你会读到来自加州大学圣地亚哥分校和麻省理工学院的计算机科学家是如何演示每个公司都需要有自己的虚拟服务器的（因此，Amazon 改变了其做事的方式）。在第二个方案中，一种新技术能将你的数据分成 16 份。当发生某类故障时，你可以用其中幸存的任意 10 份来恢复所有的数据。记住这些文章通常有一个目的是很重要的，所以你需要在确定文章中提及的产品是否满足你公司的需求时，将这一目的牢记在心中。

3.2 考虑云安全方案

如今的公司会在云端存储大量的数据以控制成本，并能从任何需要的地点访问数据。云计算，包括云数据存储，已被普遍接受，因为使用它来替代定制解决方案是有意义的。

云计算的问题在于它将你的公司暴露在各种问题面前。例如，你无从知道提供云服务的公司能否保证你的数据安全。媒体报道过太多关于公司数据被修改而导致灾难性后果的故事。事实上，你很容易就能找到关于轻松破解云存储的故事，比如 *Tech Times* 杂志上的一篇文 章“Cloud hacking isn’t as hard as most would think”（http://www.techtimes.com/articles/14800/20140903/cloud-hacking-isnt-hard-think.htm）。（一种可能解决部分云数据存储问题的方式是，在发送数据前用强加密技术对数据进行加密。当然，由于需要连续对数据进行加密和解密，这必然会影响应用程序的速度。）

下面几小节提供了用第三方解决方案去保证在线数据安全的一些做法。这几个小节考虑三个场景：数据仓库、文件共享和云存储。你可能必须组合使用几个方案来为公司创造一个

完整的计划。但是，这些方案会给你一个很好的起点，并将最终为你节省时间与工作量。最好的是，因为有其他人在维护这些方案，所以你能在长期运行中节省金钱，不需要持续修复自己定制的解决方案。



有一条要记住的经验法则是，保守秘密的最佳方式是不要告诉任何人。如果你有存储秘密数据的需求，在线保存可能是最糟糕的解决方案。无论你将数据保护得多好，如果有些人确定要窃取它，他们就能办到。破坏安全通常比构建安全要容易。所以，如果你有必须依法确保其在任何场景下都安全的数据，采用处于你控制之下的本地存储是最佳做法。文章“Are my files really safe if I store them in the cloud?”（http://computer.howstuffworks.com/cloud-computing/files-safe-in-the-cloud.htm）会告诉你黑客能用多少种不同的方式来窃取你的数据，而且可以肯定的是黑客将想出更多的方式。

3.2.1　理解数据仓库

数据仓库可以是很多东西。如果试着在网上为其找到一个一致的定义，你最终会在每个网站上都找到一个新的意思。问题在于，*数据仓库*这一术语对于每一个使用它的人来说都略有不同。尽管如此，大部分人会同意数据仓库是一个集中式的数据存储位置，用来保存作为公司知识库的一部分的信息。采用数据挖掘技术，公司能探查这个知识库并能真正从中创造新的知识。当然，数据仓库还有很多其他的含义，但最重要的一点是你在大部分情况下是在谈论大量的数据；其中一些数据仍在进行维护，还有一些数据由于历史原因而存储起来。保持这类数据的安全是一项大工程。

数据仓库比比皆是。你能在开放访问目录（Open Access Directory，OAD，http://oad.simmons.edu/oadwiki/Data_repositories）这样的网站上找到一大批开放的数据仓库，如图 3-2 所示。你可能在你的应用中使用过其中一些仓库。所以，安全不只是保持你的私有仓库安全的问题，还要确保你所使用的任何公开数据仓库也是安全的。毕竟，黑客并不在意用什么方式入侵你的应用程序（进而入侵你的公司），重要的是能够成功入侵。

很少有数据仓库只包含一个项目的数据或只影响一个项目。事实上，创建一个这样的仓库毫无意义。大部分数据仓库包含来自大量项目的数据。例如，sourceforge.net（https://sourceforge.net/）包含了 90 000 多个项目，包含了你能想到的各种编程项目，还包含一些核心软件。传统的保持数据仓库安全的方法，比如依靠管理员给服务器打补丁或管理对其的访问，在这种情况下都不能奏效，因为维护的项目太容易遭受损害了。

由安全计算机系统小组提出的、被称为安全不可信数据仓库（Secure Untrusted Data Repository，SUNDR，https://www.yumpu.com/en/document/view/34269846/secure-untrusted-data-repository-sundr-usenix）的一种新协议类型从客户端的角度着手解决这一问题。客户端可检测到文件的修改。即使服务器是不可信的或者会以某种方式受到损害，客户端仍然能检测到潜在的安全漏洞。这一系统工作的方式依赖在某个块服务器（block server）和某个一致性服务器（consistency server）上维护的日志，其采用的方法将难以（但绝不是不可能）被攻破。你可以在网页 http://slideplayer.com/slide/3389212/ 上看到该技术的一个幻灯片演示。

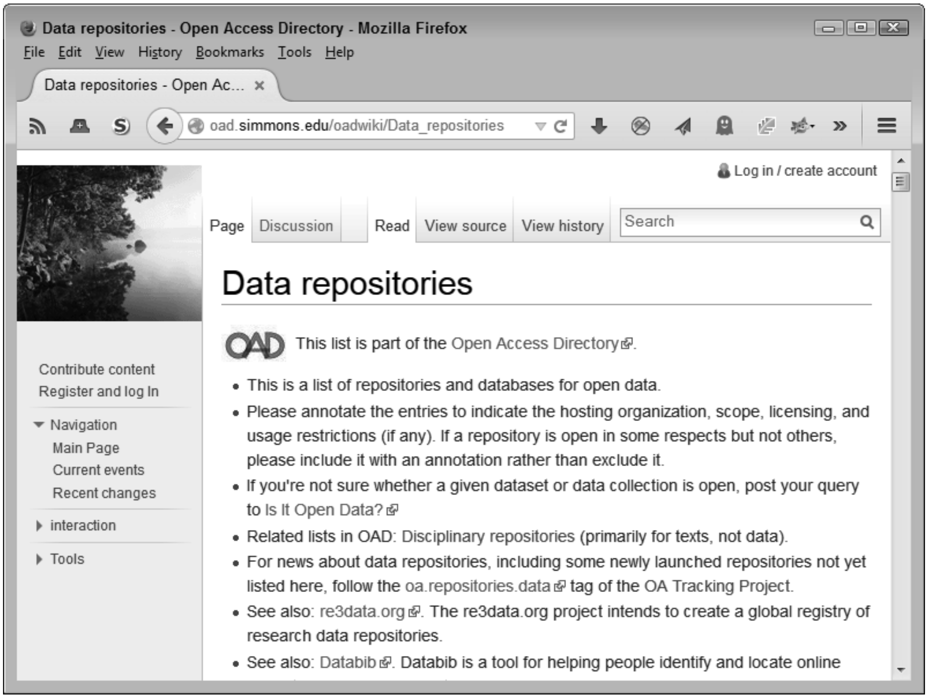


图 3-2：开放的存取仓库为某些公司提供了很关键的信息

重要的是要记住，这些新技术将与现有技术携手合作，如数据库管理器提供的安全性。它们并不会免除你作为开发者将安全纳入方案的职责。但是，SUNDR 这样的技术提供的是另一种级别的防御——一种属于服务器级别的防御，并且不依赖服务器或管理员维护。

3.2.2 处理文件共享问题

文件共享过去一直意味着在使用时要创建用户讨厌使用的繁琐配置。为了创建一个文件共享场景，公司需要搭建一个专门的文件服务器和一个访问它的 VPN。过去，对于开发人员来说，这意味着不得不写大量的代码来解决与在这样的环境下访问数据相关的问题，同时还要写大量错误捕获代码以降低设置无法正常工作时用户的挫败感。将数据从私有网络迁移到云端看起来是一个显而易见的解决方法。云安全方案由财力雄厚的第三方维护。通过将数据从私有服务器迁到云端[1]，公司能节省 65% 或更多的成本。开发人员会获得考虑周详且

注 1：“Moving your Infrastructure to the Cloud: How to Maximize Benefits and Avoid Pitfalls”（https://support.rackspace.com/white-paper/moving-your-infrastructure-to-the-cloud-how-to-maximize-benefits-and-avoid-pitfalls/）、“Cost Savings, Efficiencies Lead IT Pros to Cloud Computing”（http://searchcloudcomputing.techtarget.com/feature/Cost-savings-efficiencies-lead-IT-pros-to-cloud-computing）和“To find cloud cost savings, all you need is a little patience”（http://searchcloudcomputing.techtarget.com/feature/To-find-cloud-cost-savings-all-you-need-is-a-little-patience）这三篇文章更详细地分析了整个节省情况。

有文档的 API，这能让访问数据更加简单。用户也会受益，因为在使用可公开访问的文件共享方案时遇到的挫折会少得多，并且这样的解决方案通常能支持用户拥有的每一种设备。

关键是要认识到任何公开的文件共享服务都将制造潜在的安全漏洞。无论服务器建的墙有多高，黑客都能在下方挖个洞窃取你的数据。使用任何类型的文件共享服务带来的风险会超出你在使用 VPN 时预料到的风险。虽然许多公司都成功使用了文件共享服务，但要记住，VPN 通常会更安全，且你可能必须为某些存储需求选择一个 VPN 解决方案，虽然这会带来非常高的成本。

当今天大部分人都在考虑云端的文件共享时，你可以考虑 Dropbox（https://www.dropbox.com/business）这样的产品。Dropbox 有一个 API（https://www.dropbox.com/developers），用户可用来为自己的应用程序创建接口。该 API 提供了完整的功能，并且你能用它满足各种需求，如图 3-3 所示。当然，Dropbox 最近也占据过安全问题的新闻头条[2]，*PCWorld* 上的一篇文章（http://www.pcworld.com/article/2918524/software-productivity/how-to-make-dropbox-more-secure-without-spending-a-cent.html）详细论述了解决这些问题的方案。

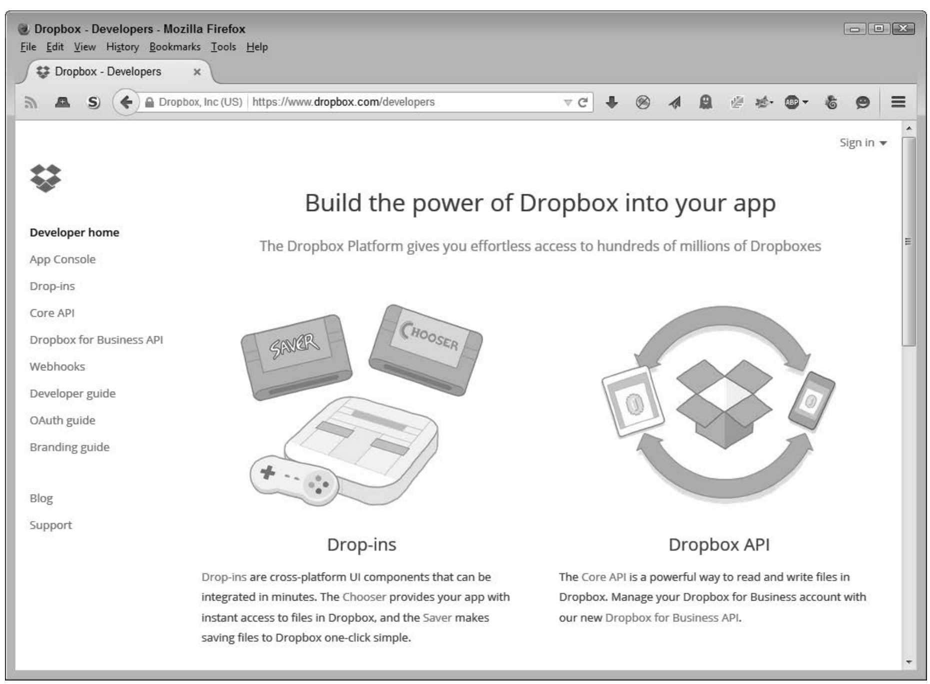


图 3-3：使用 Dropbox API 可以轻松地将在线文件共享功能添加进你的应用

注 2：例如，请参阅“Dropbox drops the security notification ball, again”（http://www.zdnet.com/article/dropbox-drops-the-security-notification-ball-again/）和“Wary of Privacy Issues? Ditch Dropbox and Avoid Google, Says Edward Snowden”（http://www.newsweek.com/wary-privacy-issues-ditch-dropbox-and-avoid-google-says-edward-snowden-276956）。

当为中小企业（small- to medium-sized business，SMB）创建文件共享方案时，你确实有很多可以尝试的解决方案。但是它们都有安全问题，所以你需要研究每一个问题以确定哪些安全问题是要优先解决的。虽然 Dropbox 这样的方案上了很多安全问题的头条，但它们也有很好地解决这些安全问题的可行方案。以下是目前最常用的文件共享方案的清单（Dropbox 除外）。

- Acronis Business（http://www.acronis.com/en-us/business/overview/）
- Box（https://www.box.com/）
- Carbonite Business（http://www.carbonite.com/en/cloud-backup/business-solutions/workstation-plans/）
- Citrix ShareFile（https://www.sharefile.com/）
- CrashPlan Pro（http://www.code42.com/business/）
- Engnyte（https://www.egnyte.com/）
- Google Drive（http://www.google.com/drive/start/index.html）
- Hightail（https://www.hightail.com/，原先称为 YouSendIt）
- Microsoft OneDrive（https://onedrive.live.com/about/en-us/）
- MozyPro（http://mozy.com/product/mozy/business）
- SpiderOak（https://spideroak.com/）
- SugarSync for Business（https://www.sugarsync.com/business）
- SyncPlicity（https://www.syncplicity.com/）

选择一个第三方文件共享方案可能非常困难。公司在选择方案时会有特定的目标，比如成本 / 收益率。但是作为开发人员，你有其他的关注点。这里列出了一些你在使用文件共享服务时应该考虑的事情。

- 有良好文档的 API

  为了写文件共享方案的自定义代码并使某些类型的访问不可见，你需要有良好文档的 API。

- 安全性

  基于云的文件共享方案需要以下这些特别的安全特性来保证其成功。

  - ♦ 云分级

    这些特性中最重要的就是云分级，你可以定义文件如何存储。有一些文件只出现在文件共享服务中，有一些除了云副本外还需要本地副本，还有一些只能出现在本地磁盘上，尽管其能通过文件共享服务访问。

  - ♦ 文件类型

    有些文件类型可能有特定的存储要求。这些文件可能包含敏感数据，且你可能需要在本地存储它们或者对它们进行加密，尽管基于文件分级的原则它们会以不同的方式出现。

  - ♦ 访问频率

    你的应用程序可能只会以某一特定频率访问文件。当访问文件的频率不一样时，可能表明出现了安全问题。文件共享服务应该就这一潜在问题向你发出警报。

- *全局命名空间*

  某些情况下，文件共享服务会提供数据孤岛（data silo）方式的访问，这样财务数据就能完全与人力资源数据隔开。数据孤岛确实能达到这一目标，但有时候数据需要以这样的方式存储：任何有权限的人可在任何地点用任何设备访问数据。要获得这种访问，你需要一个全局命名空间。

- *冗余*

  为了能安心写代码且不需要应付用户的需求，你需要有一个具备充分冗余的文件共享方案。否则，你不能确保用户需要的数据在任何时刻都能访问。但是，作为数据冗余的一部分，你需要确保数据安全地存储在每个采用单独的虚拟服务器的地方（某些类型的共享文件攻击依赖攻击者入侵目标服务器，而当文件共享服务依赖单独机器上的虚拟服务器时，这种攻击会变得更加困难）。

3.2.3　考虑云存储

*云存储*这一术语涉及大量的需求。其中有两个在本章中已专门提到：数据仓库和文件共享服务。但是，你的公司可能有其他的云存储需求。如果你能构建一个使用单一主机的应用程序，会简单得多，但这通常是不可能的。在选择主机时，你还必须考虑以下这些额外的云存储要求。

- *归档*

  数据归档不同于其他类型的云存储，你需要确保数据是安全的，而不只是可访问。如果发生重大事件，你要确保数据保持安全。公司以前会依靠保存在异地的磁带和其他媒介保证数据的安全，但如今云存储会解决这一需求。你构建的任何应用程序可能都必须归档数据，并确保以安全的方式进行归档。

- *设置信息的存储*

  用户会想要用任何设备以任何方式访问任何数据。用户不知道应用程序需要经过设置才能正常工作，并且在本地保存设置会让创建任何地方都能使用的灵活应用变得不可能。遗憾的是，保存在云端的设置也会让黑客很容易猜到应用程序的特性，进而可能入侵公司系统，所以你需要确保设置信息的存储都是经过加密的，并且是以这样的方式进行托管的，从而让任何漏洞变得明显。

- *媒体存储*

  今天的用户要求访问各种各样的媒体，包括视频、音频、图像和演示报告等。你的应用程序可能需要将媒体作为其目标的一部分，这意味着要依赖分级这样的特性来确保应用程序和云存储能正确处理数据。用户应该只能访问其所需要的媒体，而不是公司必须提供的所有媒体。

还有其他本书未讨论的云存储类型。例如，大部分开发人员对电子邮件都没有太大兴趣，除非他们正在开发一个电子邮件应用。关键是，存储涉及广泛的类型和需求，你需要将其作为安全方案的一部分去考虑。

3.3 选择产品类型

当读完本书时，你会了解如何使用不同的产品类型来让编码工作变得更简单。重要的是，要理解你可以用三种方式对代码产品进行分类：库、API 和微服务。每一种代码资源都有特定的目标并帮你满足特定的需求。在一个应用程序中混用和协调各种产品类型是有可能的，但你需要确保这些产品在使用时对安全性不会造成负面影响。接下来的几个小节将探讨不同的产品类型，并帮你理解如何将它们作为应用程序策略的一部分。更重要的是，你对每种产品类型的安全问题会有初步了解。

> **使用其他人的代码**
>
> 每次使用其他人的代码，就是在给黑客可乘之机，他们会找到方法来攻克那些代码所提供的安全措施。对黑客来说，这样做的好处是巨大的。通过制造一个安全漏洞，黑客就有可能入侵各种使用了已被其攻破的库、API 和微服务的应用程序。同一个黑客能在各处进行破坏，这意味着黑客能选择入侵哪些网站及窃取什么数据。部分黑客非常成功，他们完全控制了数据源，并将数据当作人质一样向数据拥有者勒索金钱。
>
> 使用他人代码的好处是，你可以用比平时更少的人力在更短时间内开发出应用程序。支持和维护的成本也低得多。当有人发现库、API 或微服务有问题时，进行修复的不是你，而是代码的拥有者（假设拥有者会修复；否则，你必须转而使用其他的库、API 或微服务）。不需要做任何工作，你可以自动获得由第三方提供的修复所带来的好处。
>
> 第三方也会提供一些增强，让代码更快、更有效率或更安全。当有新技术出现时，第三方会提供所需的更新，让你的应用程序以新的方式工作。如果用户需要访问新的设备类型，第三方会创建使那种设备正常运行所需的代码。总之，虽然使用他人代码的理由有很多，但这样做时必须谨慎。

3.3.1 使用库

库是存在于单独的文件中的代码，但你需要将它们加载进自己的应用程序中。库会变成你的应用程序的一部分。大部分情况下，你能把库下载到本地服务器，但在其他情况下则不能。将库下载到本地服务器的好处是，其他人看不到你如何使用它来创建应用程序。通过让加载过程更快和更简单，这一做法还能提升应用程序的速度。此外，下载库意味着库代码是稳定的，这使得有可能创建出更可靠的应用程序。

从安全的角度看，似乎将库下载到本地磁盘是最好的选择。你能获得所有使用他人代码所带来的好处，并且不需要将自己的应用程序暴露给窥伺你如何使用库的潜在黑客。从安全的角度看，缺点是使用本地副本也意味着你不能获得自动更新。这意味着你的应用程序可能会包含库开发人员已经修复的 bug，这会让黑客更容易入侵你的系统。

别人是否能看到你正在如何使用下载的库取决于库的源代码语言。通常在一个文本编辑器里就能读取到 JavaScript。为了让别人更难监测到他们在用代码做什么，很多人对他们的 JavaScript 代码进行混淆（让代码几乎不可读）。JScrambler（https://jscrambler.com/en/） 和 JavaScript Obfuscator/Encoder（http://www.danstools.com/javascript-obfuscate/）这样的产品可以让别人难以看到代码在做什么，但意志坚定的黑客仍然可以做到。反编译许多在互联网上使用的许多不同类型的代码文件也是可能的。关键是你可以通过下载库并让其变得不可读，让这样的事情变得困难，但你无法杜绝别人窥探你的应用程序如何工作。

当使用外部库时，你需要考虑原作者的声誉，以及你在应用程序中使用库的方式。因为库直接与你的应用程序交互，所以你需要考虑库应该以怎样的方式访问应用程序的内部。在使用库时，防御性地编码是很重要的，因为你不想告诉别人太多关于应用程序如何工作的信息。以下是一些外部库的例子：

- D3.js（https://d3js.org/）
- Google Web Toolkit（GWT，http://www.gwtproject.org/）
- jQuery（http://jquery.com/）
- jQuery Mobile（http://jquerymobile.com/）
- jQuery UI（http://jqueryui.com/）
- MooTools（http://mootools.net/）
- PDF.js（http://mozilla.github.io/pdf.js/）
- QUnit（http://qunitjs.com/）
- SWFObject（https://github.com/swfobject/swfobject）
- YUI Library（http://yuilibrary.com/）

3.3.2 访问API

API 是你通过向一个中心位置发起请求来从自己的应用程序访问的代码。API 以单独实体的形式存在，并且你是从自己的应用程序中对它发起请求。重点是，API 与你的应用程序的其他代码完全隔离，所以你需要创建一个对 API 的引用并发起对 API 的请求。一个 API 通常关联某种服务，比如数据存储。你可以用 API 创建一个客户端 / 服务器类型的连接，同时也会带来这种连接所蕴含的所有安全问题。

一些 API 会加入安全的访问，以帮助确保黑客不能轻易利用某些漏洞来控制 API，或以作者意想不到的方式使用它。有些 API 需要一个名称与密码的组合来访问服务，这是很好的做法，只要信息是以加密的方式发送的。但是，即使使用了加密，数据也不是真正安全的。作为一个替代方案，API 可使用密钥来提供访问。不幸的是，当密钥暴露时，API 的端点仍然会很乐意地为请求它的人提供信息。总之，API 会在某个时刻出现安全问题，但你可以用某些方法来减少黑客的活动。在使用 API 时保持警惕并观察黑客的行为也是很重要的。

API 的一个潜在问题是，它们可能会很慢。你的应用程序会变得低效并且有延迟。这是一个很重要的考虑因素，因为用户没有耐心，并且可能在等待的过程中不经意间地破坏你的应用程序。损坏的应用程序通常会表现出安全问题，黑客非常乐于利用这些问题。

当使用 API 的时候，黑客还可能采用中间人攻击的方式访问你的数据。中间人攻击特别难以检测，因为调用显然是正常的，且你不会在稍后收到的数据中发现任何异常。同时，黑客会利用收集到的数据来做某些事情，比如检索客户的信用卡号码或获取关于你的公司的有用信息。当使用 API 时，你必须密切关注数据加密等问题以确保数据安全。以下是 API 的一些示例：

- AccuWeather（http://www.programmableweb.com/api/accuweather）
- Amazon（https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html）
- Box（https://developers.box.com/）
- FaceBook（https://developers.facebook.com/）
- Flicker（https://www.flickr.com/services/developer/）
- Google（https://developers.google.com/products/）
- Pinterest（http://www.programmableweb.com/api/pinterest）
- Salesforce（http://www.salesforce.com/us/developer/docs/api/index.htm）
- Twitter（https://dev.twitter.com/overview/documentation）
- WordPress（http://codex.wordpress.org/WordPress_APIs）
- YouTube（https://developers.google.com/youtube/）

库与 API 的一个不同点是，大部分库可免费使用，但许多 API 需要支付一定费用才能使用。在大多数情况下，你可以在足以满足测试目标的级别访问某个 API，但要想获取完整功能的 API，你必须获得一个密钥，这意味着你需要为所需的访问级别支付费用。

除此之外，许多 API 供应商要求你在沙盒模式中测试自己的应用程序，并获得调试后的应用程序的证书，然后才允许你的应用程序使用 API。这些要求也增加了使用 API 的成本，这使得它们不像库那么流行。

3.3.3 考虑微服务

微服务实际上是构建于已有技术上的一种新技术。微服务是对 Web 服务、API 和库的混用，但是是以小型包的形式。事实上，微服务具有以下这些你需要考虑的特性：

- 依赖一个小型的、单一目标的、基于服务的应用程序来创建一个具有完整功能的应用程序（每个单一目标的应用程序就是一个微服务）
- 用最合适的编程语言来完成任务
- 对于具体的微服务，用最高效的数据管理技术访问应用程序数据
- 在每个微服务之间开发轻量级的通信
- 依靠 REST 等协议来通信，所以管道是哑的，但微服务是智能的
- 采取分散式应用管理，分别监控每个微服务

- 根据需要选择各种微服务来搭建任意数量的成熟的应用程序（桌面端、移动浏览器、原生移动应用，甚至是 API）

使用微服务时，你需要考虑的问题与使用库和 API 时一样。例如，微服务可能会作为你的应用程序代码的一部分，所以你需要考虑微服务如何与应用程序交互。此外，像使用 API 一样，在使用微服务时会发送和接收数据，所以中间人攻击可能会给你造成问题。

从安全的角度看，微服务实现的安全性会不同于库或 API。每个微服务会基于自身的需求提供独立的安全性，而不是为整体使用一个网站中所有微服务的每个人或事提供单一的解决方案。因此，微服务会提供更好的安全性，但这个安全也是不平衡且更难以处理的。下面是一些微服务的示例（每个网站都提供大量微服务的访问，你要在具体的应用程序中选择你想使用哪些微服务）。

- Akana（http://www.akana.com/solutions/microservices）
- Archivematica（https://ww.archivematica.org/en/）
- Gilliam（http://gilliam.github.io/）
- LSQ.io（https://angel.co/lsq-io）
- Seneca（http://senecajs.org/）

因为微服务非常新，所以你会发现很多关于微服务由什么构成的讨论。某些权威机构认为，代码行数（lines of code，LOC）是个问题。使用 200~500 行代码的服务属于微服务的范畴，但超过这一范围的服务则不是（少于 200 行代码显然不能提供足够的功能）。因此，Cloner（https://www.npmjs.com/package/app-cloner-heroku）这样的微服务就满足这一要求（305 LOC），但 Deploy Hooks（https://devcenter.heroku.com/articles/deploy-hooks）这样的微服务则不满足（1240 LOC）。

第二部分

运用成功的编码实践

在这一部分中，你将开始了解编写安全应用程序的技巧。每一章都会探讨典型 Web 应用程序的一个部分。第 4 章从 UI 开始讲起，它可能是最为重要的部分，因为安全性始于并终于与用户的配合。如果界面有缺陷，用户就不会配合（至少不会配合得太好），并且你的安全方案将会出现漏洞。第 5 章与第 4 章紧密相关。一个不受信任的应用程序对于与其交互的用户来说是令人沮丧的，且容易直接或间接产生出更多的安全问题。

现代的应用程序不会存在于真空中。如果开发人员想要从零开始构建每一个应用程序，那所有人都会遭殃。库（第 6 章）、API（第 7 章）和微服务（第 8 章）的使用让构建应用程序的过程更快，更简单。此外，由这些第三方提供的代码会接受非常多的审查，所以它们会比你自己写的代码更安全，而且，比起公司为自己写的代码，第三方提供的代码通常能更快地更新。

第 4 章

开发成功的界面

应用程序依赖界面与用户进行交互。当界面有缺陷时，用户对应用的好感会减少，且用户的满意度也会降低。你能想到的每一个安全解决方案都依赖用户的友好才得以实现。是的，你可以尝试用严厉的措施强迫用户参与，但通常这些措施会导致用户不断寻找克服安全性的方法，而不是与之配合。大部分情况下，当用户支持你时，你能得到最好的结果，这意味着你要努力创造一个用户欣赏的界面（让应用程序很易用，以至于所谓的应用程序已经从用户的视线里消失了）。当然，你确实需要确保应用程序有强制策略。因此，本章会从界面的角度同时关注应用程序安全性方面的“胡萝卜”（用户协作）和“大棒”（强制策略）。

让这个等式的“胡萝卜”部分很明显是非常重要的。当看到各种吸引人的东西、有用的提示和干净的界面时，用户会觉得开发者真的在乎这个应用以及用户如何看待这个应用。让用户体验很愉悦是很关键的。

这个等式中的“大棒”部分通常会隐藏起来或者很微妙。例如，应用程序不会让用户输入一个州的名称，而是通过创建一个州的列表并让用户从中选择，防止未经认证的输入。虽然选择受到约束，但用户看到的却是易于使用的功能。

有些界面决定甚至在编码之前就是错误的。例如，有些校验技术会告诉用户输入不合法但不告诉用户原因。更重要的是，当反馈缺少有用的提示和示例，用户会变得很沮丧。本章还讨论了可用来改善界面的技巧。有些技巧能奏效，有些则不能，但这不是什么问题，重要的是，有些技巧能减少用户的焦虑。

很少有人会再手写一个网站的代码了，明白这一点是很重要的。不用库、API、框架、微服务和任何其他第三方资源来开发一个网站的人更少。但是，所有这些第三方的源依赖本章所描述的技术。你从第三方获得的所有资源是预先打包好的代码，或许你曾自己创建过这些代码。按照本章的方式查看代码能帮助你理解第三方源所使用的底层技术，这样你能更好地辨别这些源是否为安全的。从这个意义上来说，本章的代码是设计用来帮助你理解安全性原则的，而不是你要用到自己的网站中的东西。

4.1 评估UI

大多数公司需要花费时间来评估应用程序的 UI，因为这些界面为黑客提供了明确的机会以入侵网络。1.1 节描述了黑客通常会利用的一些漏洞。大量的漏洞是依靠一些 UI 元素产生威胁的。在不影响用户交互的前提下，界面越简洁越好。接下来描述了一些可用来评估 UI 的技巧。

研究本章所描述示例的最好方式是使用可下载的代码，而不是手动键入这些代码。使用下载的代码可减少潜在的错误。你可以在下载代码的 \S4WD\Chapter04 目录下找到本章的例子。

4.1.1 创建简洁的界面

如今，创建可行界面的一个关键是让它们简洁。老式的复杂界面会制造安全问题，因为用户不确定要做什么或有太多东西要做。现在利用一些库可以创建简洁界面，这种界面在许多情况下在一个时间点只关注一个问题。例如，图 4-1 所示的选项卡文档界面就是简洁界面的一个例子，其在一个时间点只处理 Tabs.html 文件中的一部分事情。

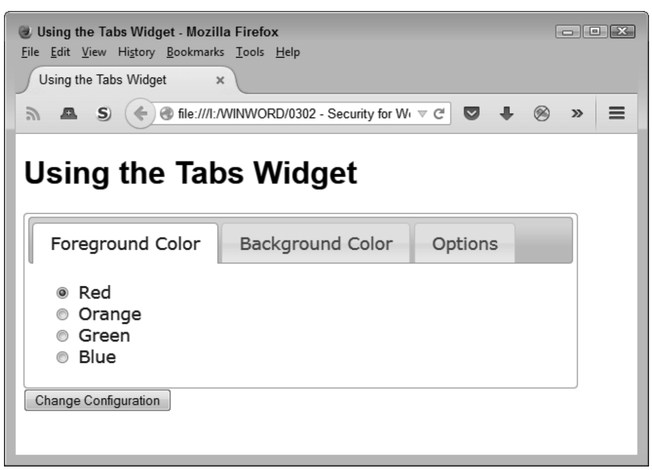


图 4-1：一个简洁的界面以用户能理解的方式聚焦于一个问题

这个界面非常简洁。它问了一个简单的问题，提供了有限数量的响应，并让用户更加难以输入坏的数据。用户选择一种颜色，点击按钮，然后移至下一个选项卡的内容。现在很多 Web 应用程序采用了这种类型的界面。它经常用于注册表单。这个例子使用了 jQuery UI 库（https://jqueryui.com/）来执行任务。下面是这个例子的源代码：

```
<!DOCTYPE html>

<html>
<head>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <title>Using the Tabs Widget</title>
   <style>
      #Configuration
      {
         width: 90%;
         text-align: center;
      }
      #Configuration div
      {
          text-align: left;
      }
   </style>
   <script language="JavaScript">
      $(function()
        {
            $("#Configuration").tabs();
        });
   </script>
</head>

<body>
   <h1>Using the Tabs Widget</h1>
   <form id="ConfigForm" method="get" action="Tabs.html">
      <div id="Configuration">
         <ul>
            <li><a href="#Tab1">Foreground Color</a></li>
            <li><a href="#Tab2">Background Color</a></li>
            <li><a href="#Tab3">Options</a></li>
         </ul>
         <div id="Tab1">
            <input id="FGRed"
                   type="radio"
                   name="Foreground"
                   value="Red"
                   checked="checked" />
            <label for="FGRed">Red</label><br />
```

```
      <input id="FGOrange"
             type="radio"
             name="Foreground"
             value="Orange" />
      <label for="FGOrange">Orange</label><br />
      <input id="FGGreen"
             type="radio"
             name="Foreground"
             value="Green" />
      <label for="FGGreen">Green</label><br />
      <input id="FGBlue"
             type="radio"
             name="Foreground"
             value="Blue" />
      <label for="FGBlue">Blue</label>
   </div>
   <div id="Tab2">
      <input id="BGRed"
             type="radio"
             name="Background"
             value="Red"
             checked="checked" />
      <label for="BGRed">Red</label><br />
      <input id="BGOrange"
             type="radio"
             name="Background"
             value="Orange" />
      <label for="BGOrange">Orange</label><br />
      <input id="BGGreen"
             type="radio"
             value="Green" />
      <label for="BGGreen">Green</label><br />
      <input id="BGBlue"
             type="radio"
             name="Background"
             value="Blue" />
      <label for="BGBlue">Blue</label>
   </div>
   <div id="Tab3">
      <input id="Sounds"
             type="checkbox"
             name="Sounds"
             value="SpecialSounds" />
      <label for="Sounds">Use Special Sounds</label><br />
      <input id="Effects"
             type="checkbox"
             name="Effects"
             value="SpecialEffects" />
      <label for="Effects">Use Special Effects</label>
   </div>
</div>
<input id="ChangeConfig"
       type="submit"
       value="Change Configuration" />
```

```
    </form>
</body>
</html>
```

为了创建这一界面，你需要引入 jQuery（https://jquery.com/）、jQuery UI 库，以及相关的 jQuery UI 样式表。选项卡文档信息出现在一个有配置 `id` 的 `<div>` 标签里。创建该界面的奥秘在于对 `$("#Configuration").tabs()` 这一语句的调用。

4.1.2　使界面灵活

削弱用户输入非法数据的能力，使界面简洁，以及保持事情简单，所有这些都以某种方式限制着用户。它们是有助于保持数据安全，同时让用户与应用程序的交互变得更简单的约束。但是，界面也需要灵活性。用户会在很多设备上使用应用程序，并不是所有的设备都能接受你提供的初始安排。让用户根据需要组织屏幕会给用户以“权力”，同时不会留下任何安全漏洞。只要有可能，一个界面应该让用户能够进行以下操作。

- 拖曳

  在屏幕上移动元素会带给用户更好的界面视觉效果，并可能减少错误的输入。至少，重新组织界面会让用户以最合适的方式与应用程序交互。

- 调整大小

  用户可能有很多理由需要调整一个界面元素的大小。可能是文字太小了。将调整大小的行为与大多数浏览器提供的根据需要使文字变大或变小的能力结合起来，有助于用户更好地浏览信息。

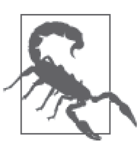
- 选择

  为已选项创建正面反馈是很关键的。你不应该对用户的发现能力做任何假设。已选项应该提供多种反馈方式，比如使用较粗的边框，以较白（较亮）的颜色呈现，更改文字属性，以及以颜色的方式提供反馈。这时，安排专人使用屏幕阅读器这样的辅助功能检查选项的效果是很重要的。

- 排序

  数据应该按照对用户有意义的顺序呈现，而不是你需要的顺序。为此，只要有可能就提供多种排序方式，这样用户能执行所需要的分析并能更容易地做出选择。

jQuery 和 jQuery UI 库提供了执行所有主要的界面灵活性任务和大量其他任务（比如允许拖放数据输入项）的方法。图 4-2 展示了使用这些库在屏幕上移动元素的一个例子。



库可以帮助你创建合适的环境以供用户输入数据。但是，这不能消除对所接收到的每份数据进行校验的需求。当开发人员基于界面假设输入时，黑客会很高兴。你应该总是假设黑客会找到绕过（你为诚实用户提供的）界面辅助功能的方法，并且要对每份数据都进行校验，无论它们从何而来。



图 4-2：让移动元素变得可能

创建一个有可移动元素的界面是很容易的。下面的代码向你展示了具体做法（你也能在 DragContent.html 这个文件中找到这段代码）。

```
<!DOCTYPE html>

<html>
<head>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <style>
      #MoveMe
      {
         border: solid;
         width: 200px;
         height: 5em;
         text-align: center;
         line-height: 5em;
      }
   </style>
```

```
    <script language="JavaScript">
       $(function()
          {
             $("#MoveMe").draggable();
          });
    </script>
    <title>Creating a Draggable Element</title>
</head>

<body>
    <h1>Creating a Draggable Element</h1>
    <p id="MoveMe">
       Moveable Paragraph
    </p>
</body>
</html>
```

在这个例子中，MoveMe 段落（<p> 元素）是移动目标。注意，让段落可移动不会影响到头部（<h1> 元素）。为了让段落元素可移动，你只需在表单加载时调用 $("#MoveMe").draggable()。

4.1.3 提供辅助功能

创建一个即使是能力最差的用户都能很好地使用的环境，对于大部分开发人员来说是很难的，因为大部分开发人员甚至无法想象不能很好地使用计算机的情况。开发人员支持的许多用户都是非常聪明的，他们只是不了解计算机或应用程序运行的方式，所以为他们提供各种各样的辅助功能是很重要的。

你通常应该提供的一个帮助项是说明文字。将说明文字添加进某一元素的 HTML 标签中。以下是一些经常用于说明文字的属性。

- title

  title 属性使得可以将说明文字添加到页面上的任何文本型或控制型元素上。当用户用鼠标悬停在有 title 属性的元素上时，浏览器会在文本气球里展示额外的信息。很重要的一点是，在 title 属性中不要使用太长的文字或除文字以外的内容，因为屏幕阅读器要依靠这些属性为有特殊视觉需要的人描述元素。

- alt

  alt 属性用于为一个图片元素提供简短的描述。当浏览器不能展示图片时，它会用展示该描述。此外，当某些有特殊需求的人看不到图片时，他们使用的辅助设施通常会用 alt 属性的文字来介绍该图片。要点就是，alt 属性提供了对图片的描述并在需要的时候作为图片的替代品。

- longdesc

  许多浏览器会忽略 longdesc 属性。与 alt 属性相比，longdesc 属性提供了对一张图片更长的描述。大部分情况下，你要将该描述限制在大约一个段落。其要点是提供足够的信息，以便让有特殊需要的人对该图片产生一个合理的心理版本，至少包括该图片的突出点（而不是背景）。

用户帮助信息有很多其他形式。大部分网站都缺失的一种帮助是提供一个示例说明：对于文本框和其他将键盘输入数据作为输入的输入框，你想要怎样的输入。你经常会看到文本框里包含了对期望内容的描述，而不是一个关于期望内容的例子（当用户在文本框里输入内容时，示例文字会消失）。例如，不要在文本框里展示“姓氏”，而是给出一个真实姓氏的例子，比如 Doe（John Doe 的姓）或类似的文字。

任何输入项还应该提供帮助链接。当用户点击该链接时，应用程序应展示有额外信息的帮助页面。这主要是要尽可能简单地为用户提供正确的信息。你从一开始就阻止的任何错误将会降低用户的挫折感，使数据输入更准确，并且最重要的是，能避免经常导致安全问题的错误。

4.1.4 定义可访问性问题

可访问性是一个解决起来很棘手的问题，特别是当使用现代工具的时候。但是，这是一个需要考虑的重要问题，因为根据 Web Accessibility In Mind（WebAIM）的数据，大约 20% 访问网站的人会有某种特殊需求：视觉、听觉、运动技能或认知。随着世界人口进入老龄化时期，这一比例会变得更大。事实上，你很有可能最终也会需要某种辅助来满足某个特殊需求。

你可能想知道可访问性与安全有什么关系。当用户对某个界面不理解时，他们就会犯错误。但是，培训通常有助于解决这一问题，因为你能教会人们如何正确地使用软件（而有时候他们确实会倾听）。当人们因某一特别的障碍而不能理解某个界面时，他们仍然会犯错误，但是再多的培训也无法解决这一问题。即使用户有积极的动机来正确地使用软件，不达标的可访问性也会阻止他们达到这一目的。错误经常会转换成最坏的安全漏洞。预想不到的输入，即你从没想过会有人提供的输入，经常造成最严重的问题。缺少可访问性的应用程序更容易创建一个糟糕的环境，从而使得破坏安全的错误经常发生。

幸运的是，一些供应商设计了具有某种级别的可访问性的库、API、框架以及微服务。例如，它们避免使用表格作为格式化工具，因为这会给那些有视力障碍的人所使用的屏幕阅读器造成问题。但是，这些工具没有一个是完美的，并且其中一些比起其他更差。因此，重要的是要测试你的网站，以确定它提供的是哪种级别的可访问性。针对如何修复在你的网站中发现的可访问性问题，大部分测试工具也提供了提示。表 4-1 给出了一份测试工具的清单以及使用方法的描述。

表4-1：可访问性测试网站

网站	URL	描述
Lynx	http://www.delorie.com/web/lynxview.html	这是一个有 Linux、OS X 和 Windows 版本但只支持文本的浏览器。它可以通过只展示文字来帮你检查网站。这个检查可以帮助你看到屏幕阅读器所看到的网站的样子，这样就更容易检测和修正问题。在 http://lynx.browser.org/ 这个网站上可以学到更多关于该产品的信息

（续）

网站	URL	描述
NIST Webmetrics Tool Suite	http://zing.ncsl.nist.gov/webmet	来自国家标准与技术研究所（National Institute of Standards and Technology，NIST）的一组工具可以帮你测试网站的可用性和可访问性。例如，网页静态分析工具（Web Static Analyzer Tool，WebSAT）能确保你的页面满足特定的可用性目标。网页变量指示程序（The Web Variable Instrumenter Program，WebVIP）能帮助跟踪用户的交互行为，所以你能知道用户找到网站功能的情况。在这个网站上还有更多的工具，并且 NIST 会经常更新它们
Opera	http://www.opera.com	与 Lynx 类似，这个浏览器能让你看到屏幕阅读器所看到的网站的样子。但是与 Lynx 不同的是，该产品还能让你开启或关闭某些特性来满足比较的需求。例如，你可以将图片特性关闭，这样你就能看到标签是什么样子（在网页 http://help.opera.com/Windows/12.10/en/images.html 上可看到这个说明）。Opera 在许多平台上都可使用，包括 Windows、OS X 和 Linux
O’Reilly XML.com	http://www.xml.com/pub/a/tools/ruwf/check.html	这个网站提供了一个 XML 语法检查器，也可以校验 XHTML。你可以直接提供一个 XML 输入或提供包含 XML 内容的 URL。该工具进行的测试会检查 XML 的格式是否良好，它不会校验 XML 实际表达的意思。大部分情况下，你会使用 W3C HTML Validation Service 来为网页做最终检查。但是，该网站会执行中间阶段的快速检查并测试 XML
W3C HTML Validation Service	http://validator.w3.org	该网站会检查你的页面的 HTML 与万维网联盟（World Wide Web Consortium，W3C）的建议和标准的相符程度。该网站上出现的错误意味着你的页面的代码是不正确的，即使大部分浏览器能处理它，所以这个测试器超越了可用性和可访问性的要求。不要以为通过这个网站的测试就能自动让你的网站具有可访问性。通过这个网站的测试只是意味着你的代码是正确的。但是，确保代码正确是确保加入可访问性特征的良好的第一步
Web Design Group HTML Validator	http://www.htmlhelp.com/tools/validator/	这个网站会检查你的页面或计算机上的 HTML。它还提供了一个选项用于选择校验单个页面还是整个网站。你可能还会发现，与 W3C HTML Validation Service 相比，该网站对其检查的页面没那么挑剔，它看起来似乎是输出一样的信息，但它可能不会提供关于网站的完整校验

（续）

网站	URL	描述
WebAIM	http://webaim.org	你可以在这个网站上找到各种有趣的工具来验证你的网站是否可访问。例如，你可以使用 WAVE Web Accessibility Evaluation Tool（http://wave.webaim.org/）来确定你的网站是否有任何可访问性问题。对于许多网站来说，颜色的使用是一个严重的问题。Color Contrast Checker（http://webaim.org/resources/contrastchecker/）这个工具可以帮助你验证那些有色觉障碍的人是否能真正看清你的网站

测试工具有助于验证开发团队在创建每个人都能使用的应用程序方面是否圆满完成了工作。但是，实际的开发工作需要一份关于要执行的任务以及要跟踪的问题的清单。网页 http://webaim.org/standards/508/checklist 上的 section 508 清单有助于你为自己的应用程序开发工作制定这样一份清单。很有趣的是，这份清单可能很难阅读，但阅读网站 http://www.section508.gov/ 上的政府文档会更难。这些要求的意义在于要确保每个人，无论他可能有什么特殊的需求，都能如你希望的那样真正使用你的应用程序。没有这些特性，你可能会发现多达 20% 的用户会遇到他们本不应该遇到的问题。

如果处理应用程序可访问性问题的时间足够长，你会知道过去出现在某些网站上的 Bobby 认可（Bobby Approved）图标。遗憾的是，该认可不再提供了，至少不是作为免费的服务。你可以在网站 http://www.bobby-approved.com/ 上找到关于 Bobby 的历史。关键是要认识到可访问性是一个非常重要的问题，而 Bobby 在帮助人们将该特性加入网站方面做了了不起的工作。IBM 最终收购了这个产品。Bobby 仍然存在，只是它不是免费的了。

4.2 提供受控制的选择

安全与实施控制有关。应用程序必须控制其与数据交互的方式。如果不破坏数据或造成数据泄露，管理数据就是创建一个安全界面的关键基础。执行控制的最好方式之一就是采用技术手段确保用户的选择仅限于你期望的选项。使用特定的输入控制会限制用户的选择，但也会让用户更有效率，因为用户可以考虑更少的可用选项。以下是一些常见的特定输入控制选择控件。

- *单选按钮*
  让用户从一些选项中选出一个选项。选择一个新的选项通常会取消之前选择的选项。开发人员最常犯的一个错误是，没有设置默认选中项。当数据在用户没有选择的情况下被发送到服务器时，服务器会接收到一个空数据并可能出现问题。
- *复选框*
  通过勾选选项，用户可以选择从无到所有的任何选项。开发人员最常犯的错误是允许冲突的选项出现。当使用复选框时，每一个选项都应该是互斥的。

- 列表框

  根据配置的不同，一个列表框可以表现得像单选按钮或复选框。列表框的一个优点是，你可以在用户还没有选择列表框控件时将它们的选项隐藏。除了开发人员在使用单选按钮和复选框时会遇到的问题，列表框还会有填充选项列表的问题。列表框作为一个控件，要让用户必须有至少一个选项可用。不可访问的数据源会给这种控件造成严重后果。

- 菜单

  根据配置，菜单可以允许单选按钮与复选框的复杂组合。当然，菜单也能用作客户端应用程序的选择组件。很多开发人员将菜单弄得太复杂。因为用户不会花太长时间搜索一个具体的选项，所以太过复杂的菜单会导致在用户沮丧时发生错误。

在创建一个受控制的选择环境时，除了标准的输入类型（包括较老的类型，比如 password），你还可以使用特殊的 HTML5 特性。但是，这种情况下的方案是为用户提供一个文本输入框，这意味着这些选项不如你想象的那样安全。以下是 HTML5 专有的控件。

- `color`

  在大部分支持的浏览器里表现为一个按钮。点击按钮会展示系统的颜色选择器。当用户选择了一个新的颜色，该颜色会出现在按钮里，这样用户就能看到当前的选择。较新版本的 Firefox、Chrome 和 Opera 浏览器都支持这个控件。

- `date`

  表现为一个日期选择器，用户可用来选择具体的日期。`min` 和 `max` 属性可让你设置可选日期的范围。较新版本的 Chrome、Safari 和 Opera 浏览器全都支持这个控件。

- `datetime`

  这个输入类型目前没有一款浏览器支持，所以你不应该使用它，即使规范告诉你这是可用的。

- `datetime-local`

  提供了日期和时间组合的控制，这样用户就能同时选择一个日期和一个时间（伴随有这些控制的限制）。当指定 `min` 和 `max` 属性时，要提供日期和时间的值作为输入。较新版本的 Chrome、Safari 和 Opera 浏览器都支持这个控件。

- `email`

  表现为一个标准的文本输入控件。但是，控件会自动校验输入以确保用户提供的是一个 email 地址。较新版本的 IE、Firefox、Chrome 和 Opera 浏览器都支持这个控件。

- `month`

  表现为一个日期选择器，用户可用来选择一个具体的月份和年份。`min` 和 `max` 属性可让你设置可选日期的范围（虽然你用的是月份，但你必须为这些属性输入日期）。较新版本的 Chrome、Safari 和 Opera 浏览器都支持这个控件。

- `number`

  允许用户输入任何数字型的值，但不允许字母或特殊符号值。设置 `min` 和 `max` 属性能控制输入值的范围。较新版本的 IE、Firefox、Chrome、Safari 和 Opera 浏览器都支持这个控件。

- range

  在大部分浏览器上表现为一个滑动条。设置 min 和 max 属性值可控制输入范围。使用 value 属性来控制初始的范围设置。step 属性可控制滑动条移动时每一个位置变化的值，这样用户就只能选择特定的值。较新版本的 IE、Firefox、Chrome、Safari 和 Opera 浏览器都支持这个控件。

- search

  使用这个 HTML5 控件没有太特殊的地方，因为它表现得就像一个标准的 <input> 标签。较新版本的 Chrome 和 Safari 浏览器支持这个控件。

- tel

  表现为一个标准的文本输入控件。但是，该控件会自动校验输入以确保用户提供的是一个格式恰当的电话号码。较新版本的 Safari 浏览器支持这个控件。

- time

  表现为一个日期选择器，用户可用来选择一个具体的时间（不是一个时区）。min 和 max 属性可让你设置可选时间的范围。较新版本的 Chrome、Safari 和 Opera 浏览器都支持这个控件。

- url

  表现为一个标准文本输入控件。但是，该控件会自动校验输入以确保用户提供的是一个格式正确的 URL。较新版本的 IE、Firefox、Chrome 和 Opera 浏览器都支持这个控件。

  > url 输入控件是目前唯一一个在智能手机上提供特殊支持的控件（在其他电脑类型里没有发现）。在某些智能手机上，浏览器会在感知到 url 输入控件时添加一个特殊的 .com 关键字。

- week

  表现为一个日期选择器，用户可用来选择一个具体的星期和年份。min 和 max 属性可让你设置可选日期的范围（虽然你用的是月份，但你必须为这些属性输入日期）。较新版本的 Chrome、Safari 和 Opera 浏览器都支持这个控件。

  > 很重要的是要记住，黑客不会遵循规则或感觉受到你的表单的任何限制。他们可能绕过这些控件提供的保护措施，以某些其他形式直接向服务器提交数据。黑客绕过保护措施的能力是你必须在服务器端对来自客户端的任何数据进行范围检查和类型检查的原因所在。

要使用这些控件，你要使用 <input> 标签。例如，要创建一个数字输入类型，你可能会使用下面的指令（详情见文件 InputTag.html）。

```
<input id="NumberInput" type="number" min=1 max=5 value=1 />
```

在这个例子中，用户会看到一个文本输入控件，但它包含一个上 / 下箭头控件，如图 4-3 所示。此外，value 属性提供了一个默认值。用户通过输入一个新的值或使用上 / 下箭头就

能轻松地更改这个值。

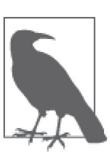


图 4-3：一个数字输入框带有一个上 / 下箭头控件

数字输入框不会阻止用户输入一个不合格的值。但是它会执行自动校验。根据浏览器不同，一个不正确的输入会以某种形式高亮提示。例如，在 Firefox 中，边框颜色会变得更浓重，且输入框会呈现红色，如图 4-4 所示。关键是，HTML5 增加的特性减少了你需要为诚实的用户所做的工作，但坚定的黑客有可能攻克它们。



图 4-4：HTML5 控件提供了自动校验



某些 HTML5 控件比其他一些更安全。例如，`range` 类型的输入框通常表现为一个滑动条，所以用户不会真的输入一个值。但是，你永远不要依靠任何控件提供绝对的安全。

4.3 选择UI的解决方案级别

当开发基于 Web 的应用程序时，你要选择实现哪种级别的 UI。事实上，大部分基于 Web 的应用程序依赖多个级别的界面控件。页面的一部分采用 HTML 标签，另一个部分采用级联样式表（Cascading Style Sheets，CSS）对界面进行操作，而第三个部分依赖 JavaScript。很重要的是要记住你能在客户端和服务器端使用 JavaScript，所以，事实上你有四种级别的解决方案可执行，如接下来的几小节描述的那样。

4.3.1 实现标准的HTML控件

4.2 节介绍了控制用户选择的方式。其中很多以 HTML 控件形式出现。HTML 控件的优点是它们在大部分地方能自动工作。浏览器必须支持控件要求的 HTML 级别，但这是唯一的要求。

不像其他大多数控件解决方案，浏览器在大多数情况下无需支持运行脚本以获得效果。当然，如果你想要完成细节的工作，就需要某种级别的脚本支持。例如，如果你想要使用特殊的控件来确保用户正确地填写了表单，可以在不使用脚本的情况下执行该任务并将表单提交给服务器处理。另一方面，如果你想要执行客户端任务，比如客户端校验，那么大部分情况下会需要脚本支持（有些 HTML5 控件能提供基础的校验支持）。

使用 HTML 控件有一些缺点。最明显的就是 HTML 控件都是基础的。如果你想要给网站添加些活力，那么 HTML 控件可能会让你失望。此外，你不能超越裸机的基础。只使用 HTML 控件不可能创建出一个良好的选项卡界面（像图 4-1 所示的那个）。

4.3.2 使用CSS控件

CSS 的核心目标是以不依赖特定浏览器、设备或平台的方式对页面内容进行格式化。此外，这种格式化不影响可访问性的需求，因为用户通常能用简单的格式替换花哨的 CSS，这样就能更好地在辅助设备上工作。也就是说，通过巧妙的编程技术，可以在各种各样的任务中使用 CSS。其中一个就是用 HTML、CSS 和 JavaScript 的组合来创建各种类型的控件，重点在于 CSS。换言之，你可以用 CSS 来创建控件，而不是依靠 HTML 或 JavaScript 来完成任务。

有可用的工具会有助于用 CSS 来创建控件。一个可查找所需工具的地方是 Dynamic Drive（http://www.dynamicdrive.com/）。当你访问该网站时，会发现各种类型的工具，包括这个例子所关注的 Button Maker（http://tools.dynamicdrive.com/button/）。图 4-5 展示了第一次访问 Button Marker 时的样子。你能使用这个页面的设置来生成创建一个微按钮所需的 CSS。但是，你可以用其他工具来简单地创建其他类型的控件。

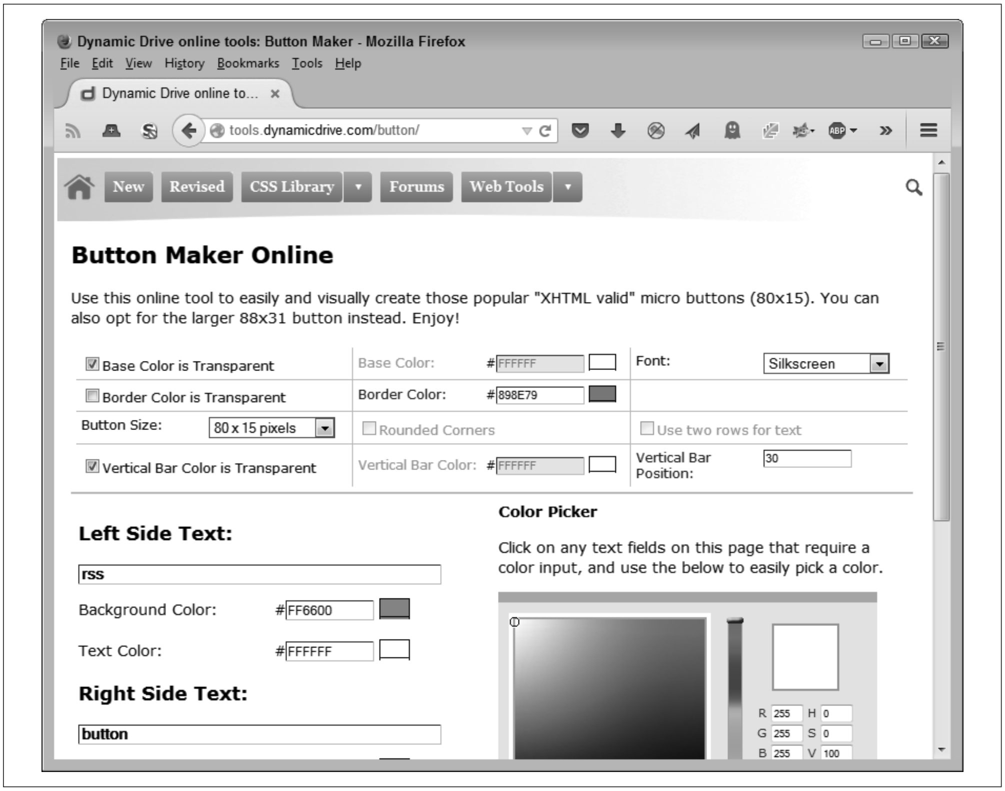


图 4-5：Button Marker 帮你创建微按钮

这个例子的输出是一些 CSS 和一个名为 MyButton.gif 的图形交换格式（Graphic Interchange Format，GIF）文件。示例代码（可在 TestButton.html 中找到）相对简单，如下所示。

```
<!DOCTYPE html>

<html>
<head>
    <title>Testing a Micro Button</title>
    <style type="text/css">
        #MicroTest
        {
            border: none;
            background-color: transparent;
        }
    </style>
</head>

<body>
    <h1>Testing a Micro Button</h1>
    <button id="MicroTest"
            onclick="alert('Clicked!')">
        <img src="MyButton.GIF">
```

```
    </button>
</body>
</html>
```

注意这段代码是如何使用 HTML 和 JavaScript 来支持的。按钮本身是一个 GIF 文件。CSS 代码执行所需的格式化任务。当然，格式化在现实的应用中会更复杂。这个例子的最终结果是一个如图 4-6 所示的按钮。



图 4-6：这个微按钮功能完整并且美观大气

创建 CSS 控件的优点是所有东西都是本地生成的，所以它们很好看，但应用程序也能执行得很好。此外，CSS 让你能够创建充满活力的应用，所以使用它们会更加有趣。CSS 的使用还能给用户视觉提示，这意味着你能用它们减少错误，即增强应用程序的安全性。

当涉及安全性时，CSS 还有某些限制。为了获得任何级别的校验，你必须使用 HTML5 和 JavaScript 技术。CSS 不能提供任何校验的可能，因为它不是设计用来执行这种任务的。记住，CSS 的设计者表示它就是用于基本的页面格式化的，除此之外没有其他的功能。通过巧妙的编程，你可以得到 CSS 能提供的一些有趣的效果。

4.3.3 用JavaScript创建控件

谈到控件，JavaScript 提供了大多数开发人员用来构建复杂应用程序的方案。是的，开发人员还依靠 HTML5 的位置支持和 CSS3 的格式化支持，但是定义界面的核心任务取决于 JavaScript。当创建一个界面时，你有两个级别的 JavaScript 支持，并且可能在一个应用程序中实现这两种级别的支持。下面描述了客户端和服务器端的控件。

1. 依靠客户端控件

本章包含了一些以不同方式使用 JavaScript 来创建客户端控件的例子。通常你不会手动创建控件，而会使用某个库，如 jQuery 和 jQuery UI。当使用客户端的控件时，应用程序从本地或远程服务器加载库，但只要库已加载，所有控件都会在本地运行。本书后面的章节（特别是第 6 章、第 7 章和第 8 章）会讨论使用进程内代码库支持的后果。客户端处理的

主要好处是你能获得速度优势，并且应用程序在多数情况下更可靠，因为你能稳定访问使应用程序正常运行的各个部分。

JavaScript 提供了客户端控件验证的最佳方案。你可以将校验配置为按你想要的方式工作。这种校验还能提供比其他类型的校验更多的功能。但是，因为黑客能看到你的代码，所以他们有可能想出攻克你的措施的策略。

使用客户端控件的一个重要的安全问题是，你正在将别人的代码融入自己的应用程序中。这意味着你的应用程序和你从来没有看过的代码之间有一个信任关系。是的，你可以查看如 jQuery 和 jQuery UI 这样的代码库的内容，但是很少开发人员有时间这样做且肯定没有耐心去检查核心的底层代码。客户端控件有潜在风险，你应该将其作为应用程序安全策略的一部分进行考虑。

2. 依靠服务器端控件

服务器端控件通常使用服务器上的某个 PHP 脚本或其他脚本启动。在许多情况下，你能直接调用脚本，传递其所需要的数据，并在浏览器中看到脚本的输出。这一过程全都在服务器上运行。因为脚本是进程外的，所以更容易保证你的应用程序与第三方代码分离，这使得你的应用程序更安全。但是当使用服务器端控件时，你要处理可靠性和速度方面的问题。

这种方案的客户端部分通常要依赖 JavaScript。事实上，通过采用如异步 JavaScript 和 XML（Asynchronous JavaScript and XML，AJAX）这样的技术，你可以将来自服务端控件的输出嵌入到页面中并根据需要更新其内容。当使用这种方式时，通过仅将数据发送给服务器并只更新该数据影响到的页面部分，可以提升应用程序的速度。

4.4　校验输入

在所有你能做的确保应用程序安全的事项中，最重要的是要假设你接收到的所有数据都是恶意的。当你假设接收到的每个输入都包含病毒或某些将破坏系统的漏洞时，就会开始以新的方式看待输入。是的，这非常偏执，但这个观念对于确保数据安全很重要。用体现恶意输入观点的方式校验数据在今天的应用环境中很重要。即便有这种观点，你也可能发现你还不够偏执（黑客在寻找掩盖恶意数据的新方法方面非常有创造力），并且需要聘请更偏执的人。接下来的几个小节会告诉你如何通过你能想到的各种方式彻底校验每一个输入，以此保护应用程序及其相关的数据。

4.4.1　只允许特定的输入

你最常接触的数据校验问题是应用程序没有很好地控制输入。当你思考这个问题时，会发现它是许多现有漏洞的问题。你永远不应该让黑客通过应用程序将一段脚本或其他不想接收的内容传递给服务器。当然，没有人会在客户端允许这样的内容，但这种内容无论如何都会来到服务器上。下面列出了一些检查方法，可用于任何从客户端向服务器传递的数据（然后要在服务器上进行二次检查）。

- 类型

  总是校验数据类型是正确的做法。字符串是最难检查的数据类型，因为根据定义，它能

包含任何东西。只要有可能就使用更严格的数据类型。例如，当你需要数字型输入时，请采用数字数据类型而不是字符串类型来传递它。

- 范围
  校验任何有范围的数据类型的范围。数字明显属于这种类型。但是，你也可以检查日期和时间的范围。
- 正则表达式
  确定数据的形式是正确的。在某些情况下，可以用正则表达式检查字符串，它将输入的字符串模式与预设的模式进行匹配。例如，当你想要一个电话号码作为输入时，它会确保你真正得到的就是一个电话号码。
- 特殊字符
  用户通常不需要发送任何特殊字符给应用程序。黑客会以非常多的方式使用特殊字符，这些特殊字符没有一个会对你的应用程序有帮助。所以，清除包含特殊字符的内容是必须要做的。

最好的用户输入是那些你从一开始就期望的输入。例如，如果你希望用户从红色、黄色和绿色中进行选择，那么可接受的答案就只能是红色、黄色和绿色。任何其他的选择都是非法的，且你应该拒绝它。你能明确更多具体的输入，就越容易保证应用程序及其相关的数据安全。

4.4.2 查找鬼祟的输入

黑客会持续寻找提供非法输入数据的方法。目的是让应用程序以某些特定的方式崩溃或迫使应用程序以意外的方式作出反应。例如，鬼祟的输入是 SQL 注入攻击的根本原因。黑客利用表单里的输入创建一个让应用程序最终以意外方式运行的条件。这种攻击造成的影响范围从损坏数据到执行不属于数据部分的脚本。你可以在网页 http://blogs.msdn.com/b/raulga/archive/2007/01/04/dynamicsql-sql-injection.aspx 和 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet 上找到关于这种攻击的一些额外信息。

> 虽然这看起来确实像个好主意，但你永远不应该允许用户提供创建动态脚本的数据。事实上，完全不使用动态脚本是很好的做法，除非你能保证用于创建动态脚本的数据源。

4.4.3 请求新的输入

黑客最喜欢利用的一个漏洞是寻求这样一些情况：应用程序会对第一次传递过来的输入数据进行严格的检查，但在随后的传递中会放松检查。由于许多语言处理循环的方式，这种情况很容易发生。只有仔细的开发者会在第二次及随后的输入重试中避免这种情况。每一次代码的输入都应该依赖相同的代码进行检查和校验输入。无论用户是第一次还是第三次输入数据，检查都应该提供相同的保护。

但是，当你开始质疑用户输入数据的动机时，这个问题有新的含义。例如，你需要考虑为什么用户需要重试 10 次才能在表单里输入正确的名字。用户知道他们的名字，这不是他们可以突然忘记的东西。在应用程序中一直重复尝试输入明显的数据可能意味着某些事情，而不只是遇见了健忘的用户。你需要质疑表单是否足够干净。当你确定表单是干净的且你有恰当的校验时，你需要开始寻找这个问题的其他原因，比如黑客正在尝试不同的方法攻破应用程序。总之，大部分应用程序应该允许一定次数的重试，然后将其关闭并等待管理员的关注。

> 永远不要假设用户经过全面的培训。有些用户需要大量的培训时间并且从未能真正准确地知道自己应该做什么。除非你想花费时间寻找那些潜伏在黑客从来不会去的地方的黑客，否则你应该将用户培训作为首要的安全需求。大量的表单重试可能指向的是更多培训时间的需求，而不是可能的黑客问题。

4.4.4 使用客户端和服务器端校验

在创建基于 Web 的应用程序时，应用程序的速度是一个首要问题。用户的关注时间在 1 到 5 秒内，不足以执行太多的校验。但是，执行校验并获得用户的正确输入是很重要的。为此，采用客户端校验看起来是可行的办法。JavaScript 程序可以快速检测错误的输入，并在数据与服务器之间来回传递之前告诉用户。事实就是，你确实需要客户端校验。

但是客户端校验有它自身的一些问题。数据必须在某些点离开其所在的系统并发送给服务器。在浏览器发送数据及服务器接收数据这段时间，黑客有各种机会拦截到数据并更改它。为了确保数据真的有正确的类型和内容，你需要在服务器端执行二次校验检查。如果无人篡改数据，二次校验将很快通过，且用户将在合理的时间内获得响应。

有些人会认为使用客户端和服务器端两层校验太过于谨慎。但是，你确实需要这两层校验。事实上，有时候无论你是否想要，你都获得了这两层校验。例如，当使用一些 HTML5 控件时，你获得了自动的校验功能。确实，校验并不总是最精确和完全的，但它为用户提供了一定级别的安全性。记住客户端校验是为了保护用户和让用户开心的。你所做的任何减少用户挫折感的事情，也会减少应用程序的支持成本和安全担忧。

服务器端的校验检查需要覆盖每一个输入和输出。只在数据从客户端到达时进行检查，然后就假设没人会在服务器上对其进行篡改是不够的。黑客会很积极地攻击服务器，因为对服务器的一次入侵可影响到大量的用户。因此，你需要对服务器上的每个环节的每个输入进行校验，以确保应用程序及其数据的安全。

你最常要处理的与校验检查相关的问题是应用程序的速度。当应用程序变慢时，并非只有用户能感受到问题，服务器负载也会增加。因此，你必须保持校验检查与速度的平衡。安全通常就是定义在追求某些目标时你愿意承受多大的风险。重要的是管理层要理解你必须维持应用程序设计与开发过程的平衡。

一些开发人员还错误地认为在某个问题上增加额外的硬件就可以解决问题。增加硬件会降低整个系统的可靠性。可靠性的降低也是一个安全风险。如果黑客有 5 台而不是 1 台服务器可攻击，那么找到一台缺少适当补丁或负载过大而足以造成崩溃的服务器的可能性会大

大提高。增加用于服务应用程序的硬件数量是一种解决方案，但你必须将其与应用程序速度的提升进行平衡，不留下重大安全漏洞。

4.5 期待意外

用户有时候是令人吃惊的。他们似乎要找到一个你没有想过要修补的漏洞。无聊的一天可能要变成一个查找是什么破坏了应用程序的会议。事实上，用户可能甚至没有去破坏应用程序；这可能只是在打电话时无意做的事情。关键是用户会找出你从未预想到的应用程序的问题。这些问题可对保护应用程序及其相关数据造成各种麻烦。有时候一个问题可使应用程序甚至是运行的服务器崩溃。你可能会惊叹于用户所做的事情，但是在现实世界里，无聊的用户会带来意想不到的事件。

黑客也会对你的应用程序执行意外的操作。但是，在这种情况下，意外是更缜密和有条理的。黑客会持续试探你的防御措施以找到入侵方法，直到有新的目标出现、黑客感到厌倦了或找到了入侵方法。大部分情况下，如果黑客确定要做，他们就能找到入侵方法。黑客要入侵你的应用程序从来不是个问题，但黑客什么时候会成功则是个问题。运用安全措施可以减缓黑客的动作，并且采用适当的监控，你就可以在黑客可能造成任何真正的破坏之前发现他们潜伏在你的系统中。

许多用户和黑客造成的意外都与输入有关。例如，你可能期望一个字符串作为输入，却接收到一段脚本。用户可以点击意料之外的组合键产生非法的字符串，从而造成应用程序或服务器崩溃。有时候我们只是假设了用户会做某件事情，但假设另一件看起来似乎更合适。例如，在要求回复数字时，用户可能回复的是一个字符串（或相反）。当然，黑客正在寻找各种恶意的方法提供你并不期望的输入，并将故意提供你不期望的输入。

第 5 章

构建可靠的代码

你可能会想，在一本以安全为主题的书中，为什么用一章来讲解构建可靠的代码。应用程序的速度、可靠性和安全性之间存在着一种内在关系。其中每个元素都在将好的应用程序变成优秀应用程序的过程中发挥一定的作用。你在强调一种元素比其他元素更重要时，一定会以某种方式削弱应用程序的重要性。当然，公司确实会有意强调其中一个元素，但这通常是为了达到特定的目标。例如，你可能有一个要不惜一切代价保护应用程序的法律要求，在这种情况下可能需要牺牲一定的速度和可靠性以达到这一目标。但是对于大部分的应用程序来说，平衡这三者是很关键的。理解速度、可靠性和安全性的关系后，你才有可能实现最大程度上的安全的应用程序，并且才有可能在三者都受到重视的环境中创建出运行良好的应用程序。

本章将可靠性视为与安全性相关的东西。换言之，本章将检验你如何平衡可靠性以达到公司具体的安全目标。本章还会将可靠性视为一门统计科学，但不会用可靠性估算的细节来烦你。重点是要理解与应用程序安全状态相关的预估风险的概念。如果有人告诉你应用程序不会产生与数据及机构完整性有关的风险，那说明这个人没有真正理解可靠性和安全性。

作为处理可靠性问题的一部分，本章将从几个角度探究可靠性。例如，为了创建可靠的应用程序，你必须定义团队协议以确保每个人都理解所需的概念。可靠性专家还要从他们进行的可靠性估算的每次迭代中学习；同样，你必须包含一个反馈回路，以公司在其特定环境中看待可靠性的方式作出改变。还有采用成套解决方案的问题。因为你不会从头开始构建很多应用程序，所以明白在应用程序中采用成套解决方案会带来的影响是很重要的。

这一章的主要思想是，世界上最可靠的应用程序是在执行任务时不会出现任何中断的应用程序。这就要排除对外部处理、输入或资源的使用，因为这三项都有故障点。这样的应用程序没有用户，且需要在可靠性极高的硬件上运行。但是，即使这些要素都满足，要想创建 100% 可靠的应用程序仍然是不可能的。每个应用程序都有可能出故障，变得不那么可靠。

缺乏可靠性会毁了公司

行业杂志的头条经常是关于安全故障的。黑客侵入所谓的安全健康记录或窃取成千上万信用卡用户的社会安全号码都是大新闻。报道可靠性的新闻似乎越来越少了。但是，缺乏可靠性会导致严重的后果，甚至是公司破产。

想想 Knight Capital Group（http://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/?_r=0）的例子。它用来与纽约证券交易所进行交互的软件出了个小差错，导致其购买了市值近 70 亿美元但其实并不想购买的股票。马上卖掉这些股票致使该公司蒙受了 4.4 亿美元的净损失。但是，损失很快变得更大。随着人们对 Knight Capital Group 失去信心，它开始亏损甚至赤字。最终，另一家公司 Getco 收购了 Knight Capital Group（http://www.reuters.com/article/2012/12/19/us-knightcapital-getco-idUSBRE8BI0OF20121219），而这全都是因为一个软件的小错误。

5.1 区分可靠性和安全性

有些人将可靠性等同于安全性。但是，可靠性和安全性是对应用程序性能完全不同的两个度量。是的，这两种度量相互影响，但许多人没有真正理解影响的方式。一个可靠的应用程序不一定是安全的，反过来也一样。接下来将详细地揭示可靠性和安全性之间的关系。

研究本章所描述示例的最好方式是使用可下载的代码，而不是手动键入这些代码。使用下载的代码可减少潜在的错误。你可以在可下载代码的 \S4WD\Chapter05 目录下找到本章的例子。

5.1.1 定义可靠性和安全性的角色

可靠性是对应用程序出现故障的频繁程度的度量。这是一个统计学度量。你可以用可靠性来回答应用程序在特定环境下出现故障的可能性有多大。可靠性还会告诉你应用环境什么时候会变化。例如，当你添加了人员且应用程序的负载增加时，可靠性就可能会降低，除非你设计的应用程序能很好地进行扩展。就算软件很完美，硬件也会到达一个极限点，可靠性仍将降低。因此，可靠性是对整体系统的衡量。硬件、用户、平台、操作环境、管理技巧以及各种其他因素都会影响可靠性，并改变你对应用程序错误或故障的看法。

安全性是对应用程序出现故障时会造成多大影响的度量。不像可靠性，没有统计学方法来

衡量安全性。你所拥有的是对破坏的可能性评估，这只能通过那些想造成破坏的黑客的决心来进行量化。如果黑客意志坚定，一定要入侵你的应用程序来制造破坏，那他们几乎就能找到方法。当处理安全性时，你还必须考虑监控的作用以及团队快速响应漏洞的能力。

在这两种情况下，当加在应用程序之上的压力大过其承受能力时，应用程序就会出现故障。出现故障的应用程序无法履行正确地管理数据这一主要职责。这种失职如何发生取决于应用程序及其出现故障的方式。可靠性故障往往导致数据的破坏，而安全性故障则往往导致数据泄露或系统完整性受损。

说明只要安全性与还要可靠性之间的区别是很重要的。以下是文件 RangeCheck1.html 中的代码，展示了客户端的安全代码（你还要在服务器端检查数据）。

```
<!DOCTYPE html>

<html>
<head>
    <title>Performing a Range Check</title>
    <script language="javascript">
        function testValue()
        {
            value = document.getElementById("Data").value;
            if (value == "")
            {
                alert("Please type a number!");
                return;
            }
            if ((value < 0) || (value > 5))
            {
                alert("Value must be between 0 and 5!");
            }
            else
            {
                alert("Value = " + value);
            }
        }
    </script>
</head>

<body>
    <h1>Performing a Range Check</h1>
    <input id="Data" type="number" value="0" min=0 max=5 /><br />
    <button id="Test" onclick="testValue()">
        Test
    </button>
</body>
</html>
```

在这个例子中，在 `<input>` 标签上使用上下箭头的用户永远不会提供限定范围之外的数据（如图 5-1 所示）。但是，`testValue()` 中的 JavaScript 代码还能确保即使是键盘输入的数据也不会超出范围。使用 `number` 输入类型意味着，如果某人输入了如“Hello”这样的值，那么 `testValue()` 实际接收到的是一个空值。它能很好地检查这种情况并为用户提供适当的反馈。

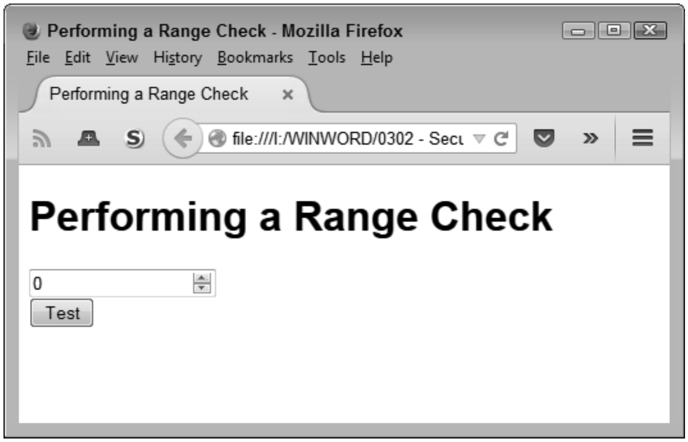


图 5-1：使用恰当的控件可让范围检查更加容易

问题在于这段代码是安全的，但不是可靠的。如果限制条件改变了，那么这段代码将不再有效。下面的 RangeCheck2.html 示例通过使用 `<input>` 标签上的 `mix` 和 `max` 属性进行范围检查，从而让代码变得更可靠。

```
<!DOCTYPE html>

<html>
<head>
    <title>Performing a Range Check</title>
    <script language="javascript">
        function testValue()
        {
            inputObj = document.getElementById("Data");
            value = inputObj.value;
            if (value == "")
            {
                alert("Please type a number!");
                return;
            }
            if ((value < inputObj.getAttribute("min")) ||
                (value > inputObj.getAttribute("max")))
            {
                alert("Value must be between 0 and 5!");
            }
            else
            {
                alert("Value = " + value);
            }
        }
    </script>
</head>
<body>
    <h1>Performing a Range Check</h1>
    <input id="Data" type="number" value="0" min=0 max=5 /><br />
```

```
    <button id="Test" onclick="testValue()">
        Test
    </button>
</body>
</html>
```

基本的检查逻辑还与之前一样。但如果有人更改了 `<input>` 标签上的 `min` 和 `max` 值，代码会通过改变检查条件自动进行响应。这段代码中的故障点比较少。

但是，要想得到既安全又可靠的代码，必须牺牲一些速度。注意第二个例子中增加的代码行数以及增加的代码调用数量。你可能不会察觉到这个例子在速度上的差异，但开始将这类检查添加到整个应用程序中时，你会看到明显的速度下降。这段代码更加安全和可靠，但用户可能会对结果不满。

5.1.2 避免可靠代码中的安全漏洞

就如安全的软件不会自动是可靠的，可靠的软件也不会自动是安全的。事实上你可能会发现，软件越可靠，其不安全的风险越高。问题在于可靠性和安全性的不同目标之一。一个可靠的应用程序通常是可访问、可使用和可预期的，这会导致更多的安全问题。以下是可靠性与安全性有不同目标的一些例子。

- 复杂的密码会让软件更加安全，但这意味着，当用户忘记密码时，软件可能变得不可用，从而使得软件不可靠。
- 确保没人篡改应用数据的检查会导致当发现篡改时软件变得不可用。
- 对安全性检查的误判会让应用程序的行为不可预测。
- 管理安全性设置会增加应用程序界面的复杂度并令其可用性降低。
- 由于安全限制（如某些政策的规定）拒绝对所需资源的访问会降低应用程序的可访问性并使其变得不可预测。

> 可靠代码中的安全漏洞所造成的后果可能是非常可怕的。在一份由 InfoWorld 最近列出的 10 个最极端的黑客事件清单（http://www.infoworld.com/article/2933868/hacking/10-extreme-hacks-to-be-truly-paranoid-about.html）中，医疗设备的事故排在第 2 位。这些设备被测试了 5~10 年以确保它们能在任何情况下持续运作。但是，没有人在测试期间对软件打补丁（涉及额外的测试）。此外，医疗设备必须易于使用，所以，在开发过程中就丢弃了任何像全面安全这样的东西。因此，通过入侵某人的医疗设备就能很容易地杀死他。在所有有严重安全问题的可靠软件中，医疗设备荣登榜首。当遭受黑客攻击时，它们也有幸成为最具破坏性的软件。

事实上，有很多安全性与可靠性严重对立的情况。你必须平衡两者，以确保应用数据保持合理的安全，且应用程序能可靠地运行。

以上一节的示例为起点，你会看到范围检查如何阻止用户输入预期范围之外的值。当然，范围检查让应用程序更安全。然而，很重要的一点是，要考虑配置范围检查的人没能正确执行任务时会发生什么，此时用户不能输入合法的值。这个应用仍然是安全的，但已变得

不可靠了。

在某些情况下，设计者可能不想要这种限制的潜在影响，并移除范围检查以使应用程序更可靠。毕竟，如果软件的目标是防止核反应堆出现危机，但软件却阻止输入能让反应堆免于出现危机的值，那软件的安全性就不再重要了，因为没有人会围绕该问题进行辩论。

处理这种情况的折中解决方案也是存在的，但它会增加软件的复杂度，并因此对可靠性造成更大影响。此外，因为折中解决方案需要增加编码，所以应用程序的速度也会受到影响。但是，在日常操作中包含各种类型的安全性检查并允许管理员重载以便在紧急时刻去除检查，用户就可以输入正确的值来挽救反应堆，但软件平时不会允许这样的输入。

关键是你通常找到的是只解决安全性或只解决可靠性难题的方案。只关注其中一个或另一个通常都不是好的做法，因为黑客（或用户）会让你在某些方面付出代价。

5.1.3 聚焦应用程序的功能

让应用程序既安全又可靠会让开发人员满怀喜悦，但用户并不在乎。用户通常将注意力放在他们关心的任务是否可以完成。用户的任务可能涉及创建一份报告。（在现实中，报告可能不是焦点，焦点可能是从投资者那里得到金钱。报告只是帮助用户达成这一目标。）如果手打报告比使用应用程序更简单且更清晰，那用户就会手打报告。用户并不在意使用什么工具来完成任务，这就是为什么你会看到用户尝试用智能手机来创造复杂的输出。作为一名开发人员，你可以悄悄地陶醉于包含在你的代码中的奇妙策略，但用户并不关心它。关键是，用户不会到你面前来说这个应用程序是不可靠的。用户的输入通常与用户的任务有关——无论任务是什么。导致用户失败的原因是否是你开发的应用程序在某些重要方面不可靠，这是由你来决定的。

当你聚焦于应用程序的功能时，可靠性和安全性之间的平衡变得更加明显。这不是简单地使任务完成的事情，而是让任务按用户最初设想的方式完成。在当今世界，这意味着要做类似下面这样的事情。

- 计数键（越少越好）
- 允许应用程序在任何平台运行
- 确保应用程序总是可用的
- 将与任务无关的步骤数减少到 0
- 使问题的答案明显，或完全避免问题

5.2 开发团队协议

朝着创建一个可靠应用程序的目标所做的任何努力都必须考虑整个团队。开发团队需要从一开始就具有设计和构建可靠应用的意识。在这个过程中，团队还需要记住保持平衡。如果一个应用程序因为运行太慢而无人使用，那将它做得既可靠又安全也没有什么意义。

团队协议会涉及各种问题。例如，在 1999 年，火星气候探测者号（Mars Climate Orbiter）在进入火星大气层时被烧毁，就是因为一组团队在使用软件时采用了公制单位而另一组却采用了英制单位（http://www.wired.com/2010/11/1110mars-climate-observer-report/）。这里的

问题在于缺乏沟通，这是你需要为成功的应用开发制定的协议的一部分。

为了给公司制定合适的协议，你需要以某些方式将任务分解。开发应用程序的团队必须从以下三个不同的级别考虑可靠性的问题。

- 偶然发生的设计或执行错误

  当考虑可靠性问题时，大部分人会考虑导致应用程序以偏离开发团队最初的设计方式运行的错误。应用程序不能正确地执行任务。但是，这些错误还可能导致诸如为黑客入侵打开缺口或不能提供所需灵活性的后果。通过采用开发人员培训和安全开发训练，以及使用专门设计用来定位这种可靠性问题的工具，开发团队可以解决这个问题。

- 更改技术

  应用程序在你完工的那天就变得过时了。事实上，有时应用程序在开发完成之前就已经过时了。技术的变更不利于软件，使其变得不可靠。以下两种级别的更改是你必须要考虑的。

  - 面向未来的

    为了创建一个能让应用程序保持其技术优势的环境，你必须令其面向未来。实现这一目标的最好方式是，将应用程序做成是由很多可交互组件组成的，但它们也是独立的实体，这让你可以升级其中一个组件而无需升级整个系统。这是微服务变得如此受欢迎的原因，但你也可以用一体化设计来实践模块编码策略。

  - 黑客的改进

    黑客会进行创新并将他们的工作做得更好。一个安全性或可靠性问题在你刚开始项目时可能还不是一个问题，但在你完成整个应用程序之前可能就变成一个问题了。在黑客指出它们之前，你可能甚至都不知道有问题存在。解决这一问题的最好方式是更新你的工具和技术以对抗黑客的改进。

- 恶意

  如果有些人在你的开发团队里待得不开心或带着寻找软件漏洞的目的做着公司的某项工作，那他很可能会给你造成破坏。这名团队成员甚至可能引入故障或后门程序，以便在离职之后利用这些漏洞。当然，你不想要营造一种监视的气氛，因为这样做会扼杀创新且会造成更多的问题。但是，你还是需要确保所有的管理人员真的在管理开发团队。

团队成员之间的沟通是很重要的，但确保沟通没有被误解则更加重要。想当然的假设制造了各种问题，而人们特别擅长用假设来填补信息鸿沟。当然，一名团队成员的假设可能不是由同一个团队的其他成员来执行。开发团队所需的沟通应包括以下这些最佳实践［你可以在 Cyber Security and Information Systems Information Analysis Center 的网站（https://sw.csiac.org/databases/url/key/2）上找到更多最佳实践、工具和资源］。

- 可靠性和安全性培训

  除非团队成员说相同的语言且对可靠性概念的理解在同一水平上，否则他们无法沟通。达到这一目标的唯一方式是，确保团队成员都接受了适当的培训。在创建一个培训计划时，请确保培训是一致的，无论你使用的是外部还是内部的培训师。

- **可靠性要求**

  开发一个应用程序但没有首先定义你想要什么，这就好比在没有蓝图的情况下建造房子。就如同手里没有蓝图时你不会开始为造一个房子而挖地基一样，在没有首先明确可靠性和安全性在应用程序整体运行中的角色之前，你不会开始任何编码工作。确保你制定的任何定义都包含针对每个开发阶段的具体指标和目标。这些定义还应该概述安全性与可靠性检查的使用、代码审计与测试。

- **可靠的设计**

  正如在开始项目时要识别应用程序会面临的安全威胁一样，你还必须识别可靠性问题并提出解决方法。设计过程必须包含如何处理可靠性问题的具体细节。虽然很多公司意识到需要有安全性专家来帮忙解决安全性问题，但很少有公司意识到存在类似功能的可靠性专家，比如可靠性咨询服务（http://www.reliasoft.com/consulting/）。

- **可靠的编码**

  编码时必须将安全性和可靠性牢记在心。除非将所学和所设计的东西投入实践，否则你做再多的展示也没有用。

- **安全源代码处理**

  为了处理恶意破坏等威胁，公司必须进行安全源代码处理。这意味着只有接受过适当培训和受认证的人才能看到源代码。此外，这还意味着你要执行设计与代码的检查，以确保应用程序还忠实于最初的设计目标。

- **可靠性测试**

  测试代码以确保其真正达到可靠性目标是很重要的，这一目标是应用程序需求与设计方案的一部分。可靠性测试会包含各种问题，比如当有负载时或失去对某一资源的访问时应用程序会如何响应。这个任务是测试应用程序的故障点，并验证其是否能成功地处理每个问题。

  > 盲测试（让没有使用经验的组外人员尝试使用该应用程序）能够得到一些有趣和有用的、关于应用程序潜在安全漏洞的信息。这个测试常常能披露出组合键使用的问题，黑客会利用其入侵应用程序。这个测试还能展示出应用程序的功能不能自解释的那些区域，以及需要返工、额外的文档或培训的地方。

- **可靠性和安全性文档**

  用文档记录需求、设计、编码技巧、测试技巧和其他用于确保应用程序可靠且安全的过程是很重要的。该文档应该说明你是如何找到平衡点的，并解释你如何将它们作为应用程序要求和设计的一部分进行处理。

- **可靠性和安全性准备**

  在发布应用程序之前，开发团队需要确保没有新的威胁出现在应用程序必须处理的场景中。可靠性和安全性都会涉及风险，而这个阶段决定了新的威胁所带来的风险。作为软件更新的一部分，处理低风险的威胁是可行的，不需要影响应用程序的发布。

- 可靠性和安全性响应

  在发布应用程序之后，开发团队需要及时地对任何新的可靠性和安全性威胁作出响应。很重要的一点是，要理解开发团队以外的来源可能会报告这些问题，且开发团队被期望能提供快速的响应。

- 完整性检查

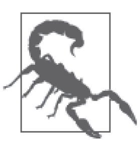
  确保应用程序按预期方式持续运行就意味着要采用某些签名技术（以确保没人可以修改代码），从而让代码安全。此外，开发团队应该持续测试代码以应对新的威胁，并验证应用程序是以安全的方式在管理数据。其思想是要持续查找潜在的问题，即使你肯定没有问题存在。黑客很希望看到你的团队不爱检测新的威胁，直到一切都变得太迟了。

- 安全性与可靠性研究

  为特定的人员分配任务，让其在那些新威胁出现时找到它们并提出处理方法，这是很重要的。对应用程序进行测试是好的，但知道如何测试其是否能应对最新的威胁是确保测试有效的唯一方式。

5.3 创建经验教训的反馈回路

可靠性基于随时间推移的事件的统计分析。时间过得越久且记录的事件越多，预测就会越准确。故障事件间的平均时间被称为平均无故障时间（mean time between failures，MTBF）。大部分软件代码似乎并没有从这个角度探讨该主题，但像任何其他东西一样，软件有故障模式，且可以通过分析这些模式创建出一张能让你预期软件在什么时候会出故障的蓝图。当然，像任何统计一样，这不可能确定精确的时间点，而只能确定在特定环境下特定应用程序活动的一般过程。



有些人并不认为 MTBF 是衡量软件可靠性的正确方法，因为他们觉得 MTBF 只是指出下一次软件 bug、环境问题或其他因素导致软件故障的时间。记住，MTBF 是基于某一特定环境的。因为软件要在不同的公司中运行，而不同的公司很少有完全相同的运行环境，所以试图创建出一个能在各家公司工作的应用程序的 MTBF 是不可行的。针对某一公司的某一应用程序的 MTBF 是可以的，因为该公司的环境不太可能会发生巨大的变化。当环境变化时，MTBF 的值就不再有效了。

一个应用程序的 MTBF 越高，支持它所花费的时间就越少，维护成本也越低。此外，高 MTBF 值也预示着由软件故障所导致的数据泄露也会更少。应用程序的故障不只是因为软件 bug 才会出现，还会由于错误的用法、环境问题（比如内存不够）、不可重现的原因（比如宇宙射线导致的能量峰值）以及其他原因而出现。因为不可能管理所有这些故障原因，所以软件从来不会是 100% 可靠的。在某种程度上，即使是最好的软件也会遇到故障。

> 宇宙射线真的会影响计算机。事实上，计算机存储所在的纬度越高，受到的影响就会越大。芯片中晶体管的大小也会影响软件错误的发生率。你可以在文章“Cosmic rays and computers”（http://www.nature.com/news/1998/980730/full/news980730-7.html）、“Effect of cosmic rays on computer memories”（http://www.ncbi.nlm.nih.gov/pubmed/17820742） 和“Should Every Computer Chip Have a Cosmic Ray Detector?”（http://www.newscientist.com/blog/technology/2008/03/do-we-need-cosmic-ray-alerts-for.html）中发现更多关于这一有趣影响的信息。这些信息很有趣，且可能最终解释了一些你曾经看到过的不可重现的错误。

对故障原因的一个反馈回路能帮你提高某一公司中的 MTBF。通过测试故障原因，你可以制订一个计划来提升 MTBF，以降低可能的成本。以下的事项在分析软件故障原因时进行考虑。

- *质量*

  当软件质量提升时，MTBF 就会变高。查找和移除 bug，改进 UI 以减少用户犯错，添加检查以确保所需资源在使用之前是可用的，这些都可以提升软件的质量。

- *故障点*

  减少应用程序故障点的数量会提升 MTBF。减少故障点的最好方式是通过精简程序和提升它们的效率来让应用程序变得简单。当然，你还可以在可能的时候做增加冗余这样的事情。例如，拥有两份相同的数据源可以降低一份数据源丢失后应用程序出现故障的可能性。

- *培训*

  更好的培训会减少操作失误并提升 MTBF。现在有一种“培训的回报在减少”的观点，而如今的大部分用户对于他们工作要用的软件都没有接受足够的培训。此外，培训支持人们更准确地处理故障且让开发人员发现故障的真正原因，这将有助于改进反馈过程。

创建完整的故障及其原因列表是开始理解应用程序动态性的最好方式。随着对某一应用程序基础知识的增加，有可能发现一些预测模式，并且能使用这些模式来确定在哪里花费时间和资源来让进行修正。当然，有些故障原因不可能得到解决。是的，你可以屏蔽所有硬件以摆脱讨厌的宇宙射线，但公司花费这种钱的可能性是极低的（并且回报率很小）。

5.4 考虑成套解决方案的问题

如前面的章节所提到的，如今大多数应用程序依靠成套的解决方案来执行普通的任务。想要完全从头开始搭建一个应用程序要耗费过多的时间和高昂的费用。没有理由重复别人做过的工作。但是，使用成套解决方案会影响应用程序的可靠性。你依靠别人写的代码来让自己的应用程序工作，所以那部分代码也是可靠性评估的一部分。接下来会探讨你在使用成套解决方案时需要考虑的一些问题。

5.4.1 处理外部库

外部库带来了许多有趣的可靠性问题。最重要的问题是应用程序通常会在每一次启动运行时下载库的一份副本。如果你让应用程序持续运行，这一过程不会经常发生，但大部分基于 Web 的应用程序是在客户端系统中运行，这意味着客户端需要在每一次启动应用程序时去下载库。由于库的下载，速度成了一个问题。但是，如果某些因素阻止了客户端成功下载库代码，应用程序甚至可能无法启动。例如，考虑一下使用 jQuery UI 来创建一个如图 5-2 所示的折叠效果。

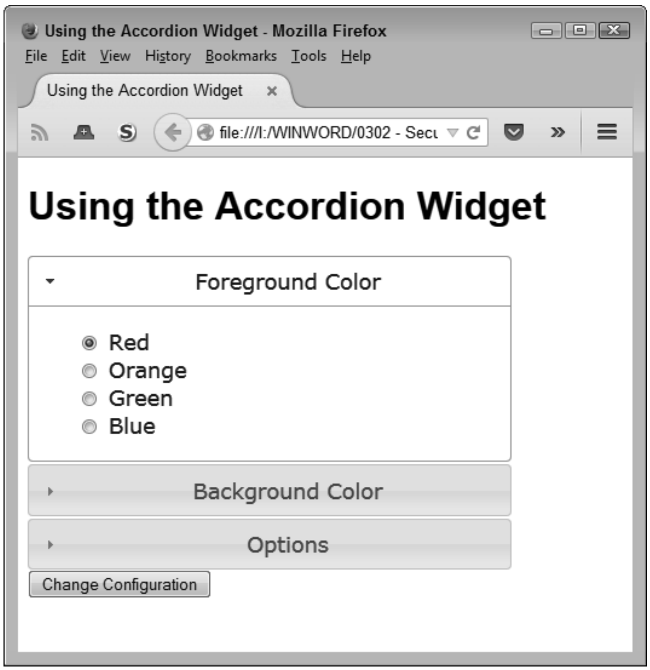


图 5-2：就算是一个简单的 jQuery UI 的例子也要求下载代码

当这个例子所依赖的库文件由于某些原因变得不可访问时，这个例子甚至无法启动。接下来的代码（可在文件 Accordian.html 中找到）会出现在 HTML 的头部以创建所需的连接。

```
<head>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
```

如果该例子三个文件中的任何一个缺失了，应用程序都不能运行。这个例子还展示了选择

库的两种方法。注意，jQuery 库依赖最新的版本，这意味着你能自动获得 bug 修复与其他更新。但是 jQuery UI 库（及其相关的 CSS 文件）都依赖某个特定的版本。在这种情况下你不会得到更新，但如果已经加入了临时修补的代码，那么或许可以修复造成应用程序崩溃的 bug。

提升使用外部库可靠性的最好方式之一就是将库下载到本地磁盘并使用这一副本，而不是直接使用供应商网站上的副本。这个策略能确保你一直使用相同的版本并降低库变得不可用的概率。当然，你需要提供本地磁盘空间以保存这些库，但与所获得的稳定性相比，增加的本地资源开销的代价很小。

与所有其他事情一样，使用外部库的本地副本也有代价。最重要的一个代价是缺少升级。大部分供应商会提供相对频繁的升级，包括 bug 修复、可靠性提升和速度提升等。当然，库通常还会包含新的特性。与新特性相对的是那些你可能需要用来让应用程序运行的旧特性。为了获得可靠性，你需要放弃某些潜在的安全性提升以及其他你可能真正想要在应用程序中使用的更新。

一个折中的方案是将库代码下载到本地，持续跟踪库的更新，并为应用程序安排更新的时间，以便让你所使用的主要库与所需的安全及特性更新保持一致。这就意味着要按你的时刻表执行更新，而不是按供应商的时刻表。虽然这一方案看起来是一个完美的解决方案，但其实并不是。黑客常常依靠零日（zero-day）攻击对最多数量的应用程序造成最大的伤害。因为你的应用程序不会自动更新，所以你还要面对发生灾难性的零日攻击的可能性。

5.4.2　处理外部API

外部 API 提供了使用外部代码访问数据及其他资源的方法。API 通常是你拿来实例化为对象并用于发起调用的类的集合。这些代码并不在客户端系统中运行，而是在服务器上以一种进程外（out-of-process，OOP）的远程过程调用（remote procedure call，RPC）方式执行的。当客户端对服务器发起请求时，一个客户端 / 服务器的请求 / 响应循环会发生。创建一个与 API 的连接跟创建一个与库的连接非常类似，只是你通常必须提供某种类型的密钥来获得访问，如 GoogleAPI.html 文件中所示的那样（可在第 7 章中看到该例子的更多细节）。

```
<head>
  <script type="text/javascript"
   src="https://maps.googleapis.com/maps/api/js?key=Your Key Here&sensor=false">
```

这个查询要求客户端构建一个请求并发送给服务器。在使用 Google API 时，你必须提供界面经纬度等数据项，以及你想要的地图类型。API 会直接将数据写入到请求数据中的应用程序位置上，如以下这段代码所示。

```
// 这个函数会在屏幕上显示地图
function GetMap()
{
   // 构建发送给Google的参数列表
   var MapOptions =
   {
      center: new google.maps.LatLng(
                 Latitude.spinner("value"),
                 Longitude.spinner("value")),
```

```
        zoom: 8,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }

    // 向Google提供地图要在屏幕上展示的位置以及地图的参数项
    var map = new google.maps.Map(
        document.getElementById("MapCanvas"),
        MapOptions);
};
```

该代码的调用结果就是一幅地图会展示在客户端的画布上。图 5-3 展示了该应用的输出的一个典型例子。

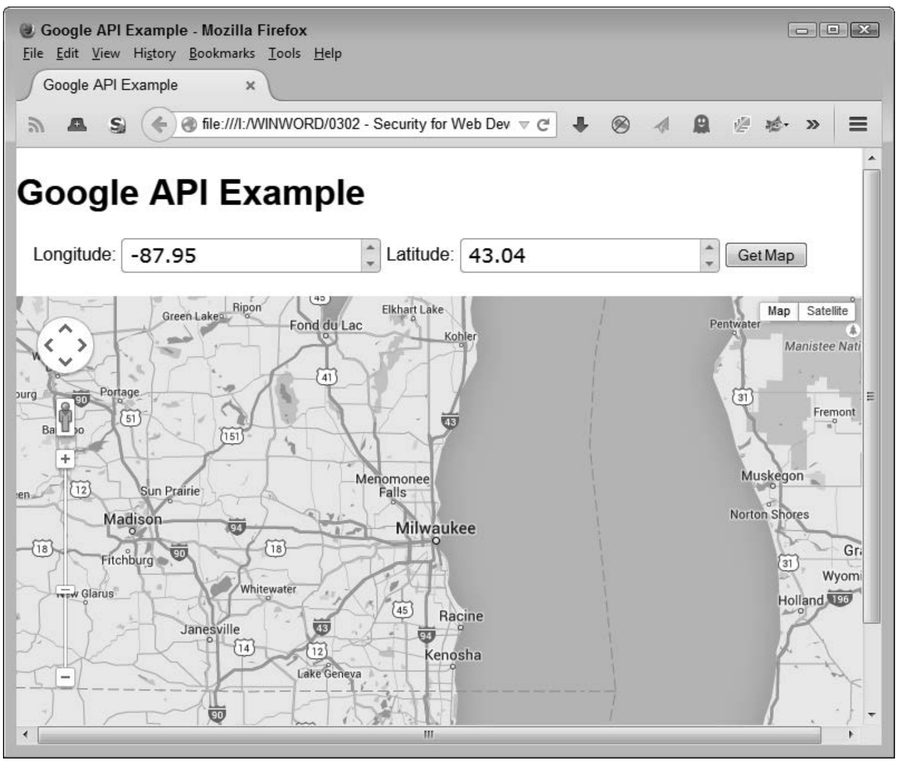


图 5-3：Google API 直接在客户端画布上画出地图

使用外部 API 或许可以使得在用别人代码时有较少的安全问题和速度问题（取决于请求数据是否比下载代码快）。但是从可靠性的角度看，外部 API 也有各种风险。任何连接中断都意味着应用程序不能工作。当然，对于开发人员来说，创建补偿这一问题的代码并向用户展示一条消息是相对容易的。比较不容易解决的是连接速度慢、数据处理以及其他能反映出连接状况的问题，或者是正在工作的黑客。在这种情况下，用户的挫败感会成为一个问题，因为用户会为了获得某些应用程序响应而提供一些错误的输入，从而让情况变得更加糟糕。

解决这种情况下的可靠性问题的一个方法是为调用计时。当应用程序检测到调用花费的时间比平时长，它就有可能给出一些反馈，比如告诉用户连接比较慢、联系管理员或以某些其他方式对问题作出反应。关键是要确保你能跟踪每次调用花费多长时间并恰当地采取行动。

5.4.3 使用框架

框架表现出的可靠性介于库和 API 之间。大多数情况下，框架是压缩的代码。一个最常使用的框架是 MooTools（http://mootools.net/）。但是还有许多其他的框架可用，你需要找到适合自己需求的那一个。在大部分情况下，框架能在服务器、客户端或两者结合的环境中运行。依据自己使用框架的方式，你会发现某些可靠性问题，这些问题同样出现在使用库、API 或两者的组合时。

区别框架和库

虽然框架［比如 Dojo（https://dojotoolkit.org/）］和库（比如 jQuery）看起来很相似，并且你以大致相同的方式使用它们，但 Dojo 是一个框架而 jQuery 是一个库。从安全性、可靠性和速度的角度来看，这两者都是不同的。

库是纯粹的代码，可以下载下来作为应用程序的一部分运行。你可以直接调用函数，且这些函数的源码有时候是可以直接获得的，这样你就能更改函数的行为。

框架提供了一种与行为交互的方式，这意味着某些任务对于开发人员是不可见的——开发人员请求框架执行任务，而框架决定如何完成它。代码仍然可以从供应商下载且仍然会成为应用程序的一部分，但底层的技术是不一样的。有些人将框架定义为经过包装的库，它提供结构和代码。

MooTools 等产品有趣的地方在于你可以执行许多与使用库相同的任务。例如，你可以用 MooTools 创建另一个版本的折叠示例。下面的代码很简单，但指出了要点。

```
<!DOCTYPE html>

<html>
<head>
   <title>MooTools Accordion Demo</title>
   <script src="MooTools-More-1.5.1.js"></script>

   <style>
      .toggler
      {

         color: #222;
         margin: 0;
         padding: 2px 5px;
         background: #eee;

         border-bottom: 1px solid #ddd;
```

```
      border-right: 1px solid #ddd;

      border-top: 1px solid #f5f5f5;
      border-left: 1px solid #f5f5f5;

   }
</style>

<script type="text/javascript">
   window.addEvent('domready', function()
   {
      var accordion = new Fx.Accordion('h3.atStart', 'div.atStart',
         {
            opacity: false,

            onActive: function(toggler, element)
            {
               toggler.setStyle('color', '#ff3300');
            },

            onBackground: function(toggler, element)
            {
               toggler.setStyle('color', '#222');
            }
         }, $('accordion'));
      });
   </script>

<body>

   <h2>MooTools Accordion Demo</h2>
   <div id="accordion">

      <h3 class="toggler atStart">Section 1</h3>
      <div class="element atStart">
         <p>Section 1 Content</p>
      </div>

      <h3 class="toggler atStart">Section 2</h3>
      <div class="element atStart">
         <p>Section 2 Content</p>
      </div>

      <h3 class="toggler atStart">Section 3</h3>
      <div class="element atStart">
         <p>Section 3 Content</p>
      </div>
   </div>

</body>
</html>
```

你可以从网页 http://mootools.net/more/builder 上获得文件 MooTools-More-1.5.1.js。请确保你包含了核心库。Google 在网页 https://developers.google.com/speed/libraries/ 上提供了一个

代码库托管网站，但它不包含额外的功能，比如 `Fx.Accordion`。

该框架要求你设置一系列标题和分割符以包含折叠的内容，如例子中的 HTML 部分所示。当被选择或未被选择时这些元素项如何出现取决于你设置的 CSS。该脚本定义了两个事件：`onActive`（当该项被选择时）和 `onBackground`（当该项未被选择时）。在这个例子中，MooTools 表现得非常像一个库，你可以看到如图 5-4 所示的输出。

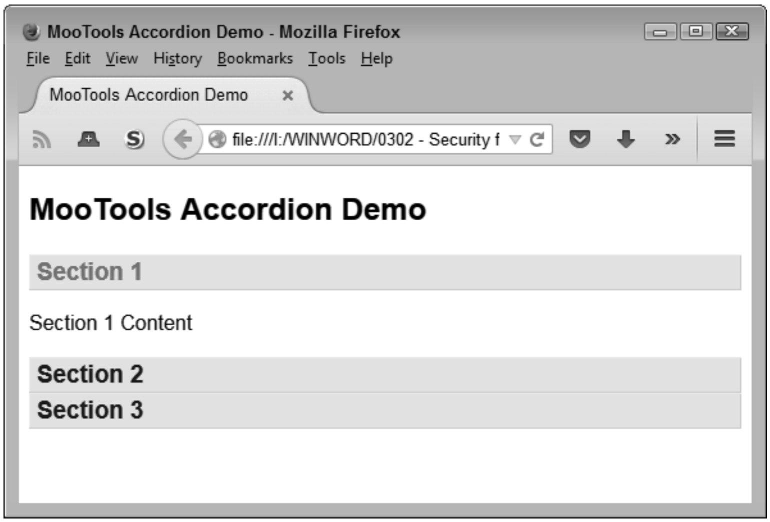


图 5-4：MooTools 是一个提供了库和 API 功能的框架

5.4.4 调用微服务

在许多方面，一个微服务就是一个细粒度的 API。使用较少的代码就有可能依靠任何所需的资源去执行给定的任务，微服务的这一理念是很有意义的。从可靠性的角度看，由于微服务的工作方式，你所知道的使用 API 的很多问题也会出现在微服务上。例如，调用某一微服务可能会花费比预期长的时间，从而引起用户的挫败感。解决这一问题的方法是提供应用程序的超时机制。但是，就微服务而言，你通常可尝试对可用的备选源发起调用。

但是，微服务在某些方面还是与 API 不同的，因为前者比较小且更加可定制化。为此，以下列出了你在使用微服务时需要考虑的一些可靠性问题。

- 使用更多不同源的微服务会增加连接失败的可能性并降低应用程序的可靠性。
- 选择能真正优化实现特定需求所需的服务的微服务，这可以提升应用程序的可靠性，因为这会降低出现重大错误的风险。
- 依靠一致的接口来进行微服务调用会减少潜在的代码错误，增强应用程序的可靠性。
- 拥有多个提供相同服务的微服务源会减少单个故障点的数量，提高应用程序的可靠性。
- 使用小型的服务意味着单个服务的丢失不会让所有服务都不可用，但使用 API 就会，所以这种情况下其可靠性较高。

第6章

包含库

很难找到一个不依赖库的 Web 应用程序，即使能找到，那也不是很重要的应用程序。使用库是不重新定义轮子的典型。事实上，与轮子类似，库有各种不同的形状、大小和颜色。是的，当你考虑众多包含可用于装饰网站的主题样式的库时，甚至颜色也会开始起作用。因此，你很可能已经在使用库方面有很多积累。事实上，本书前面章节的许多例子都依赖库，就是因为从头开始重写代码并没有意义。

本章不会试图向你灌输使用库的价值，你应该已经明白了其中的价值。但是，你可能没有考虑过在应用程序中使用别人代码的各种后果。使用库总是有一些后果的。我们的目标是安全地使用库，从而可以获得使用别人代码的所有好处，同时不会在自己的应用程序中制造任何重大的安全漏洞。（请放心，你肯定会制造某些安全漏洞；这不可能避免。）

获取最佳的库运行速度

你可能发现某些网站告诉你完全不要使用库，因为它们会导致速度等方面的问题（例如，参阅 http://www.giftofspeed.com/dont-use-javascript-libraries/）。事实是，库确实会拖慢网站，特别是当你没有明智地使用它们时。速度问题足以严重到使得用户点击跳转到下一个网站，这意味着你不能为访问网站的用户提供服务或其他资源。

本书主要使用库的未压缩版本，以方便你使用例子，并且更轻松地执行任务，比如使用调试器检查应用程序。但是从速度的角度来说，你通常需要使用库的压缩版本。压缩版本包含相同的代码，但它加载得更快，并且有时还能加快调用速度。

还有一点很重要，要认识到你不会使用库中的所有代码。jQuery UI（http://jqueryui.com/download/）等很多库提供了一个构建器，让你可以创建个性化版本的库。在明确你会使用库中的哪些部分之后，使用构建器生成只包含那些部分的版本。这个库会加载得更快，且能减少库的攻击面，这意味着安全性的提升。

6.1 考虑库的使用

使用库的方式会直接影响安全的程度，至少在某种程度上是这样的。例如，比起使用库格式化页面，直接使用自己的 JavaScript 代码介入库的操作会带来更多潜在的安全漏洞。当然，库的任何用法都会带来潜在的安全漏洞，这会消耗公司的时间、精力与资源。接下来将探讨你可能会如何使用库来执行各种任务，并考虑哪种形式的用法会造成某种类型的安全漏洞。

> 研究本章所描述示例的最好方式是使用可下载的代码，而不是手动键入这些代码。使用下载的代码可减少潜在的错误。你可以在下载代码的 \S4WD\Chapter06 目录下找到本章的例子。

6.1.1 用库增强CSS

CSS 用于格式化页面内容。基于 CSS 当前的状况，你可以惊奇地发现可添加到网站中的各种活力。Web 开发人员使用 CSS 来让枯燥的页面变得出彩。当然，CSS 也有普通的用法。例如，想象一下，在不使用 CSS 对内容进行格式化的情况下，尝试阅读有多列布局的文字。你最终得到的是一个混乱的结果。单纯使用 HTML 标签格式化内容会使页面在当今可能使用的各种设备上几乎不可用。

只使用 CSS 就可能创造出精彩的页面。但是，很重要的是要认识到，就算是最优秀的 Web 开发人员也没有时间去手写所有的 CSS 代码。有很多把 CSS 代码塞进库里面的方法，其中比较有意思的一个是 jQuery UI 提供的 `css()` 函数，如在文件 Transform.html 中可找到以下代码。

```
<!DOCTYPE html>

<html>
<head>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <script
      src="jquery.transform.js">
   </script>
   <link
      rel="stylesheet"
      href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <title>jquery.transform.js Demonstration</title>
   <style type="text/css">
      #TransformMe
      {
         border: double;
```

```
         border-color: Blue;
         border-width: thick;

         position: absolute;
         width: 120px;
         height: 40px;
         top: 90px;
         left: 75px;
      }

      #TransformMe p
      {
         margin: 0;
         padding: 0;
         height: 40px;
         font-family: "Comic Sans MS", cursive, sans-serif;
         font-size: large;
         color: Red;
         background-color: Yellow;
      }

      #Rotate
      {
         position: absolute;
         top: 190px;
         left: 75px;
      }
   </style>
   <script type="text/javascript">
      $(function()
      {
         $("#Rotate").click(function()
         {
            $("#TransformMe").css("transform", "rotate(45deg)");
         });
      })
   </script>
</head>

<body>
   <h1>jquery.transform.js Demonstration</h1>
   <div id="TransformMe">
      <p>Some Text</p>
   </div>
   <div>
      <input type="button"
             id="Rotate"
             value=" Rotate 45 Degrees " />
   </div>
</body>
</html>
```

这个例子中的功能依赖脚本 jquery.transform.js。但是，在你深入研究脚本之前，要注意该页面依赖标准 CSS 来摆放内容。在大部分情况下，你创建的应用程序会依赖本地代码与库

代码的组合。这个例子的内容包括了一个作为标签的段落和一个按钮。

点击按钮会触发 `click` 事件，它会执行一行代码。`TransformMe` 的 CSS 被修改为包含一个 45 度的旋转。文本从平直变成有一个角度，如图 6-1 所示。

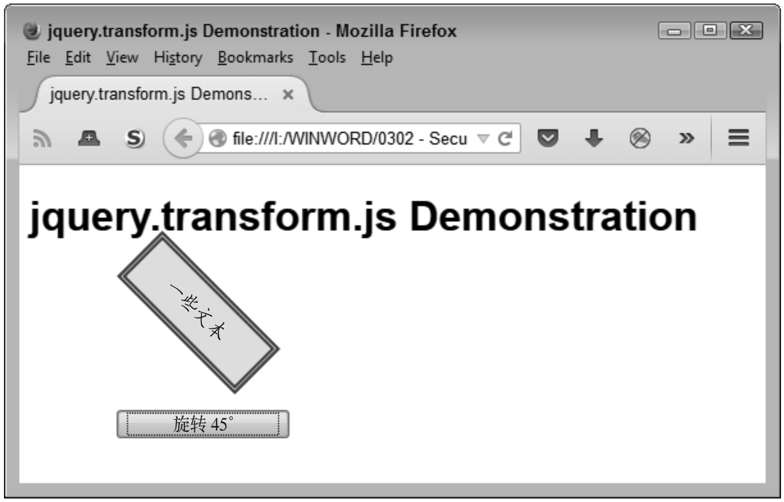


图 6-1：使用旋转能更改屏幕上文本的表现

允许脚本影响页面格式的问题在于，你不能确定脚本执行完之后页面会变成什么样。页面会包含隐含的元素，或根本不展示想要的信息。隐藏元素可能包含将用户重定向到受污染网站的代码，或执行与应用程序无关的行为。

6.1.2 用库与HTML交互

有些库会以两种常见的方式直接与应用程序的 HTML 进行交互。第一种是与已存在的元素进行交互。在这种情况下，内容通过库函数的调用出现在元素里面。第二种方式是创建新元素并将它们添加到已存在的文档中。新元素可以包含各种内容，包括可能对最终用户造成麻烦的某些标签。

使用库直接与网站的 HTML 进行交互的主要问题在于，库会给元素填充各种误导性、不正确或被彻底污染的数据。例如，它没有创建你所期望的链接，而是将用户重定向到另一个网站，从而可能会将病毒或其他恶意软件下载到用户的机器。操作 HTML 的库会导致你甚至不能马上知道的一些微小问题，因为一切看起来都在正常工作。除非用户投诉了相关的错误指向（并根据 URL 异常的工作方式），否则你不会知道相关的问题，直到很多机器受到影响进而导致服务器出现问题。

将新元素添加到页面是一个很普遍的操作。在文件 ViewJSON.html 中可找到基于名为 Test.json 的 JSON（JavaScript Object Notation）文件将新元素添加到文档中的例子。接下来的代码展示了这个例子如何执行任务。

```
<!DOCTYPE html>

<html>
<head>
   <title>Viewing JSON Files</title>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script language="JavaScript">
      function ViewData()
      {
         // 从硬盘上获取数据
         $.getJSON("Test.json",
            function(data)
            {
               // 创建一个数组来保存数据
               var items = [];

               // 遍历每个用户对象来解析数据
               $.each(data.Users,
                  function(key, value)
                  {
                     items.push("<li>" +
                        value.Name + "<br />" +
                        value.Number + "<br />" +
                        (new Date(
                           parseInt(value.Birthday.substr(6)))
                        ).toDateString()
                        + "</li>");
                  });

               // 将结果放在一个无序列表中
               $('<ul/>', {html: items.join("")}).
                  appendTo('body');
            });
      }
   </script>
</head>

<body>
   <h1>Viewing JSON Files</h1>
   <input id="btnView"
          type="button"
          value="View JSON Data"
          onclick="ViewData()" />
</body>
</html>
```

在这个例子中，库打开文件 Test.json，并从中读取数据。这段代码移除了 JSON 文件格式并添加了 HTML 标签格式。然后使用 `join()` 函数将新元素添加到页面中。图 6-2 显示了这个例子的输出。

注意，这个例子避免了 `document.write()` 调用的使用，这个调用会让用户收到各种不想要的内容。使用 `join()` 函数可以提供更为安全的方式，它能让页面远离安全问题。

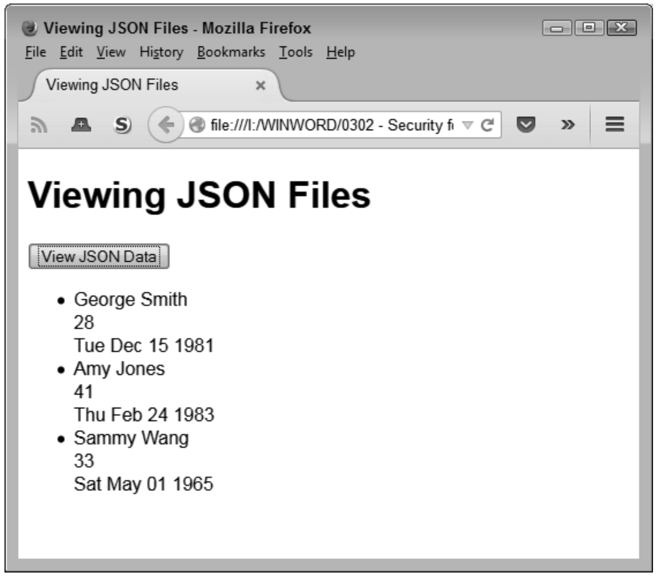


图 6-2：注入外部数据源的数据通常会带来问题

6.1.3 用库扩展JavaScript

大部分人考虑的是库扩展 CSS 或 HTML 的方式。信息的展示是最前沿的。但是，库也会影响底层的脚本。例如，大部分人熟悉如何使用 `alert()` 来展示简单的消息。大部分库都会包含某些内容，黑客可以利用其在屏幕上展示其他信息或以其他方式影响应用程序。例如，文件 DialogBox.html 中包含一个创建自定义弹框的例子，如下所示。

```
<!DOCTYPE html>

<html>
<head>
    <title>Creating a Simple Dialog Box</title>
   <script
      src="https://code.jquery.com/jquery-latest.js">
   </script>
   <script
      src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
   </script>
   <link
      rel="stylesheet"
      href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
   <style type="text/css">
    .Normal
    {
        font-family: Arial, Helvetica, sans-serif;
        color: SaddleBrown;
        background-color: Bisque;
    }
```

```
    .Emphasize
    {
        color: Maroon;
        font-style: italic;
        font-size: larger;
    }
   </style>
</head>

<body>
    <h1>Creating a Simple Dialog Box</h1>
    <div id="DialogContent"
         title="Simple Dialog Example"
         hidden>
        <p class="Normal">
            This is some
            <span class="Emphasize">interesting</span>
            text for the dialog box!
        </p>
    </div>
    <script type="text/javascript">
        $("#DialogContent").dialog();
    </script>
</body>
</html>
```

在这个例子中，当你打开应用程序时会立刻展示一个弹框，如图 6-3 所示。这种功能带来的威胁是，用户打开应用程序想要某种体验，却得到完全不同的东西。因为弹出框作为应用程序的一部分出现，所以用户会提供弹框要求的各种信息或执行其让用户执行的各种操作。

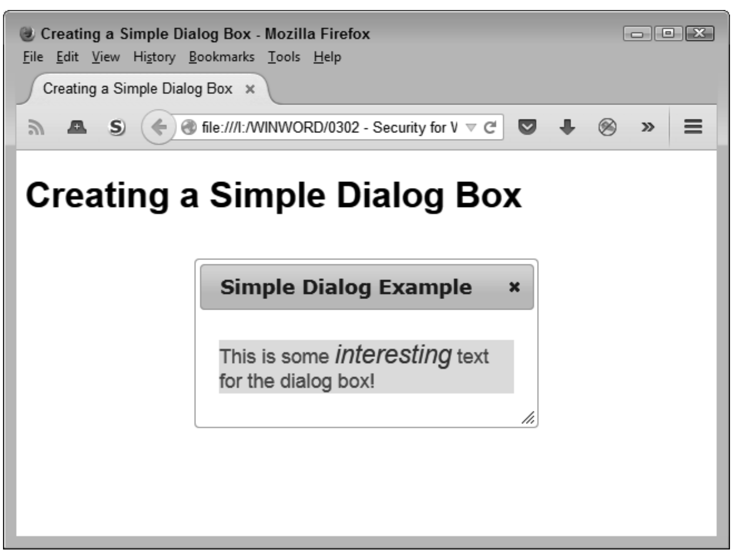


图 6-3：jQuery UI 的弹窗功能可用于制造一个安全漏洞

黑客可以通过脚本（代码）注入等方式启用这一功能。例如，黑客可将代码隐藏在博客页面的评论里（这就是为什么人工审核评论是一个好主意）。这其中的攻击是一门社会工程学，通过将弹框作为应用程序的一部分进行展示会让用户觉得放心，进而利用这一情况做些恶意的事情。

6.2 区分内部存储库和外部存储库

你可能会认为所有的 JavaScript 库都存在于第三方网站上。确实有很多流行的 JavaScript 库（如 jQuery）存在于别人的网站上。但是，很多公司也有自己的私有库。影响内部存储库的因素有以下这些。

- 它们存在于由公司拥有且维护的系统上。
- 公司控制着源代码并能根据需要修改它，以解决开发人员碰到或由别人引起公司注意的问题。
- 外部人员在没有公司授权时不能访问这些库。
- 与这些库的连接依靠内部专线，而不是通过互联网访问。

内部存储库有很多保存在第三方网站的外部存储库所没有的优点。最主要的是，在相同的编码水平下，内部存储库会比外部存储库更快、更可靠且更安全。（例如，黑客可以轻易下载存储在外部网站上的库的副本，但对于内部存储库则不太可能做到这一点。）此外，由于库只包含了公司真正需要的代码，因此很可能内部存储库会使用更少的资源，这是因为其比通用版本要小。当然，以上对内部存储库做了很多可能不对的假设。以下是一些你需要考虑的点。

- 构建和维护内部存储库成本很高，所以公司可能不会持续更新以保证安全。
- 很少有公司能组建一个在技术水平上与第三方供应商相当的开发团队，所以库的质量可能会有问题。
- 第三方库会接受大量测试人员的测试，所以即使是很小的错误也能被发现。内部存储库通常只接受了少量的测试，甚至可能没有排除一些重大缺陷。

内部存储库可行的关键在于，其目标是一些特定的功能，而第三方库不能提供这些功能，并且开发团队将它们组合起来需要很大的精力。当综合考虑应用程序的设计时，必须权衡使用内部存储库所带来的优点和风险。此外，必须进一步定义第三方库所带来的风险。

6.3 定义库带来的安全威胁

你所创建的任何 JavaScript 代码都可能包含某些黑客可加以利用来达到各种目的的错误。库也不例外。但是，因为你正在将别人创建的代码包含进自己创建的应用程序中，所以就有可能出现平时运行良好的代码由于双方的假设而最终出现了安全缺陷。为此，每个开发人员在使用库时都需要知道的两个工具是，专门为安全需求设计的测试工具，以及一位优秀的安全专家。

> 你维护的库不可能是完全安全的。你可以说自己编写了满足安全性最佳实践的代码，并且安全专家也已经检查过代码，但就算有最好的意识和可靠的工具，你也不可能保证一个库（或任何其他代码）是安全的。这一问题在非程序员（特别是公司的管理人员）想要使用你的库完成开发任务时频繁出现。每个人都清楚地知道你已经采取了各种恰当的安全措施，但没有人能保证所有代码都是完全安全的。

即使经过了测试，你也必须意识到，在库中、库所支持的应用程序中以及这两者的组合中，可能还存在着安全漏洞。为此，下面列出了一份你在使用库时最常遇见的威胁清单（在具体的缺陷来源上没有偏重）。

- *跨站脚本*（XSS）

  开发者最常面对的安全问题是 XSS。以下三种简单的方法可以处理 XSS。

  - 永远不要在同一个 HTTP 响应中传递不受信任的数据（如 HTML 或 JavaScript）。事实上，如果能让主 HTML 文档保持静态是最好的。

    > 你可能想知道服务器最终会如何将不可信的数据传递给用户。这比你想象的要容易。文章“ScriptGard: Automatic Context-Sensitive Sanitization for Large-Scale Legacy Web Applications”（http://www.comp.nus.edu.sg/~prateeks/papers/scriptgard-ccs11.pdf）描述了保持数据的完整性有多难，即使你使用了正确的净化器。安全的假设是，任何不是你自己创建的数据都是不可信的。

  - 当你必须将不可信的数据从服务器传递到客户端时，确保你以 JSON 格式对其编码，并且数据有值为 `application/json` 的 `Content-Type`。
  - 一旦准备展示数据，请使用 `Node.textContent()`、`document.createTextNode()` 或 `Element.setAttribute()`（只需要第二个参数）这些调用以确保页面可以正确地展示它。

- *危险的函数调用*

  JavaScript 支持某一调用并不意味着这一调用是安全的。使用 `setInnerHtml()` 或 `.innerHtml =` 可以注入你不想要的脚本代码。使用 `setInnerText()` 则不会。

- *直接修改 document*

  虽然你可能在本书中看见过几次 `document.write()`，但这只是为了方便，在生产环境中使用这一调用会带来灾难。这种代码可在任何地方写任何东西。请使用那些添加、删除或更新 DOM（Document Object Model，文档对象模型）元素的调用。

- *动态创建脚本*

  任何时候你把字符串转变成脚本，你就正在邀请黑客来提供这些字符串。使用 `eval()`、传递字符串参数的 `setTimeout()`、传递字符串参数的 `setInterval()` 或者 `new Function()` 会让你的代码严重不安全。

- *能执行但会导致安全缺陷的代码*

  有些 JavaScript 调用会给你各种绳子让你把自己吊起来。为了避免这种情况，使用

JavaScript 的严格模式可确保只有安全的调用可以工作。

- 与声明不符的内容

  黑客喜欢发送给你与你所想不一样的内容。避免这一问题的最好方式是遵循一个内容安全策略，这意味着要包含合适的内容标签，比如 `script-src 'self'` 和 `objectsrc 'self'`。

> 使用如 JSLint（http://www.jslint.com/）等工具扫描库可帮你确认代码质量。高质量的代码不太可能包含导致安全问题的错误。但是，很重要的是要认识到，JSLint（或类似的工具）不会专门扫描安全问题。事实上，你不能在代码中扫描安全问题。为了检查安全性问题，你必须开始专门为此测试你的代码。如果代码要求比测试所能提供的更高等级的安全保证，那你还必须聘请安全专家来人工检查代码是否有潜在的问题。

6.3.1 启用严格模式

较新的浏览器提供了对 ECMAScript 5 的支持，这包含了 JavaScript 的严格模式。为了使用这一安全特性，你需要确保用户不再使用自己的机器上的化石级浏览器。例如，IE9 就不支持这一功能，但 IE10 支持。你可以在网页 http://kangax.github.io/compattable/es5/ 上找到哪些浏览器支持严格模式。很关键的是，你要制定一个组织策略，要求用户使用特定的浏览器以使应用程序得以工作。

严格模式对 JavaScript 的操作方式做了许多明显的和微小的改变。重要的是使 JavaScript 应用的调试变得更简单，因为你的 JavaScript 应用程序如今会在遇到问题时报错，而不是悄悄地失败并表现出怪异的行为。这意味着过去常常无故死掉的奇怪的库调用现在会告诉你相关问题的信息。以下是严格模式可帮你解决的主要问题。

- 消除对 `with` 的使用

  使用 `with` 语句会给应用程序带来安全问题。最新版本的 JavaScript 已经丢弃了这一功能，因为它太危险了。一旦你尝试在代码中使用 `with`，严格模式就会报错。

- 防止错误变量

  JavaScript 的一个主要问题是，通过给一个写错名称的变量赋值，你可以突然创建出新的变量。在某些情况下，这种错误会创建出导致安全漏洞的不正确的全局变量。严格模式会强制你在使用每一个变量之前先声明它，这样就不再可能突然创建出变量。

- 不允许强制多态使用 `this`

  有些已存在的代码会强制局部变量变成全局状态，在其处于未赋值或 null 状态下时给它们赋值。例如，你不能在通过调用 `new` 来实例化一个对象之前，在构造函数中给一个变量赋值。

- 阻止重名

  开发人员很容易在对象中创建重复的属性或在函数里命名相同的参数。如果你尝试这样做，严格模式就会报错。

* 对不可变值的改变会发出通知

  在 JavaScript 里要改变一个不可变值是不可能的。但是在过去，这一尝试只是会悄悄地失败，所以有可能你认为代码当前处于某一状态，但实际却处于另一状态。严格模式会在任何尝试改变不可变值时报错。

严格模式适用于 ECMAScript 5 及以上版本。你不需要安装任何特别的东西来获得严格模式的支持。但是你必须包含 "`use strict`"; 这样的字符串。较旧版本的 JavaScript 会忽略这一字符串，所以你不会发现旧代码的运行有什么改变。文件 StrictMode.html 中包含下面的例子。

```
<!DOCTYPE html>

<html>
<head>
    <title>Working with Strict Mode</title>
    <script language="javascript">
        function testStrict(Name)
        {
            "use strict";

            try
            {
                this.Name = Name;
                alert(this.Name);
            }
            catch (e)
            {
                alert(e);
            }
        }
    </script>
</head>

<body>
    <h1>Working with Strict Mode</h1>
    <button id="Test" onclick="testStrict('George')">
        Test
    </button>
</body>
</html>
```

如果没有严格模式检查（你可以简单地把相关语句注释掉试试），这段代码会展示错误赋给 `this.Name` 的值 George。但是，有了严格模式检查，你会看到如图 6-4 所示的错误消息。

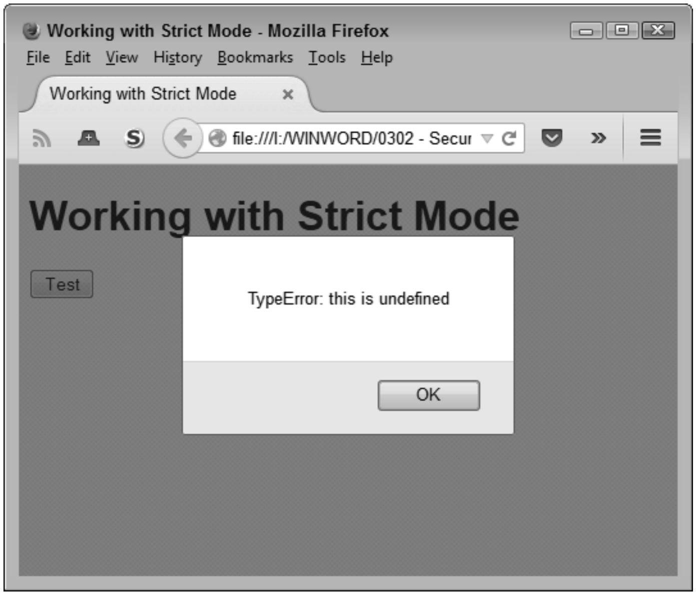


图 6-4：严格模式检查会阻止不正确的赋值

> 永远不要全局使用严格模式。一开始这似乎是一个很好的主意，但你很快会发现某些库不能运行。严格模式背后的思想是，确保你对自己开发的代码有严格的控制。使用校验检查，你可以监控你所使用的库的行为。

6.3.2　开发CSP

开发人员面对的最大问题之一是，要开发不容易受到各种脚本或 XSS 等内容的注入攻击的应用程序。由于浏览器的工作方式，这一问题并不容易解决。浏览器实际上信任来自某一特定来源的任何内容，所以浏览器会信任与元数据处于同一级别的任何注入的内容。内容安全策略（Content Security Policy，CSP）旨在通过创建白名单这类措施来解决这一问题。

对于支持 CSP 的浏览器来说（可在网页 http://caniuse.com/contentsecuritypolicy 上找到列出可兼容浏览器的表格），CSP 提供了让浏览器免于识别来自不受支持的网站的脚本及内容的方法。该策略表现为加在页面顶部的各种头部信息。当浏览器看到这些头部信息时，会使用它们来确定哪些脚本和内容是可以安全加载的。

> 当浏览器不提供对 CSP 的支持时，它会忽略你提供的头部信息并表现得跟平时一样。这意味如果你想要将 CSP 作为阻止错误内容的方法，就必须依靠支持 CSP 的浏览器。否则浏览器将跟平常一样工作，并继续加载来自不受支持的网站的内容。

头部包含三个基本要素：策略名、数据类型和数据源。下面是一个只包含一个数据类型的CSP头的例子。

```
Content-Security-Policy: script-src 'self'
```

在这个例子中，CSP声明浏览器应该只执行嵌入到本页面的脚本。如果你还想提供对jQuery的支持，就需要为此扩展策略，如下所示。

```
Content-Security-Policy: script-src 'self' 'https://code.jquery.com'
```

你不需要提供源代码文件的具体位置，只需要主机站点的信息。每一个入口通过空格与下一个分隔开。一个CSP可以定义各种内容类型。例如，你可能决定你要支持在本地页面中找到的脚本，但你根本不想支持对象。在这种情况下，你要使用下面的CSP头。

```
Content-Security-Policy: script-src 'self'; object-src 'none'
```

注意内容类型通过分号（;）分隔。CSP支持许多不同的内容类型指令。你可以在网站http://contentsecurity-policy.com/上找到一份包含这些指令列表、样例值和描述的快速指南。重要的是要记住，使用CSP可以让你避免代码中的黑客可加以利用的微小漏洞。

6.4 安全地包含库

有些公司简单地将一个库加到它们的工具箱里，因为其提供了所需的功能，而没有仔细考虑应如何将库与其他代码集成在一起，甚至没有考虑使用它是否是一个好的做法。至少在某些情况下，那些没有仔细考虑库的使用的公司最终都会后悔，因为这些库包含了大量的安全漏洞，运行得不可靠或很慢，或者与从一开始就自己开发库相比，其要耗费公司更多的时间。接下来的几个小节将帮助你确定将有趣的库包含进应用程序中是否真的是一个好主意。

拥抱沙盒方案

即使你检查了第三方的JavaScript代码并完全清除了所使用的数据，某些东西仍然很有可能会悄悄泄露。要记住，推倒一面墙要比建造它更容易。在谈到JavaScript的安全威胁时，你必须想到每一次有新的防护措施时，黑客也会用新的方式来绕过你的安全措施。

解决这一问题的一个潜在方式是依靠沙盒。本质上来说，这意味着第三方库在比自己创建的代码较低的权限级别上运行。使用沙盒不能确保应用程序是安全的，但它降低了应用程序出现安全漏洞的可能性。你可以在网页http://www.slideshare.net/phungphu/a-twotier-sandbox-architecture-for-untrustedjavascript上读到关于沙盒技术的信息。幻灯片里描述的技术都很复杂，但它们都是有效的。与所有事情一样，最终都归结于你在运行一个应用程序时能够承受什么级别的风险。第10章探讨了沙盒使用的细节。

6.4.1 充分研究库

任何时候你选择将一个库添加进自己的应用程序中，就展示了对库作者的信任。即使有称职的安全专家的帮助，将一个库拿出来并对其每一处的代码进行测试（假设能得到完整的代码）会花费比从头开发库更多的时间。（确保你拿到的不是压缩版的库，不然会很难检查，因为供应商移除了空白字符。）因此，你必须考虑将这个库作为你创建的应用程序解决方案的一部分，带来的风险是否足够小。当研究一个库时，考虑一下以下问题。

- 行业杂志是否以负面的方式讨论过这个库？（如果这个库有已知的未解决的安全漏洞，你可能需要重新考虑是否将其用于自己的应用程序中。）
- 是否有其他公司使用了该库且没有问题？
- 开发这个库的公司在回答关于库的完整性问题时是否是积极的？
- 创建这个库的公司是否提供了一致的 bug 修复和更新？
- 人们在这个库的在线支持中问了哪些问题？
- 这个库存在了多长时间？（存在时间很长的库通常问题更少，并且创建者的支持工作也做得更好。）

6.4.2 精确定义库的使用

没有人会使用应用程序中每个库的每个功能。有些研究网站宣称开发人员只会使用库的一个或两个功能。用文档记录你要如何使用一个库，可将注意力聚焦在相关部分，从而帮你规避使用库所带来的风险。更重要的是，跟踪具体的使用部分能告诉你何时必须执行一次更新，或如何通过移除用不到的功能来缩小库的规模。对于如何使用库来执行特定的应用任务了解得越多，就越能降低使用它时的风险。

6.4.3 保持库的小规模和内容聚焦

只要有可能，就应将第三方库保持在小规模且聚焦于特定的需求。这意味着你要移除你不会使用的代码，因为这些代码会制造以下问题。

- 拖慢下载速度
- 增加安全问题
- 降低可靠性
- 导致不必要的更新

幸运的是，存在一些最好的第三方库。它们想让你创建一个定制化版本用在自己的网站上。例如，当使用 jQuery UI 时，你可以使用特定的构建器创建一个定制化版本，如图 6-5 所示。

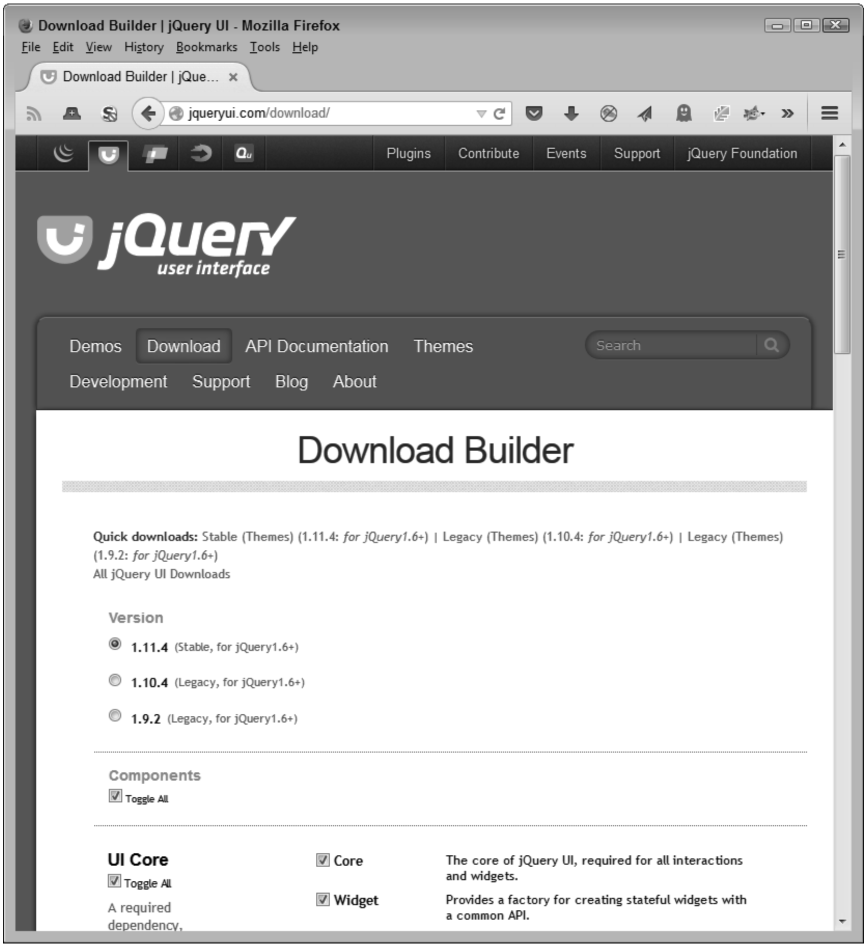


图 6-5：使用 jQuery UI 构建器可帮助你创建一个定制化版本的库

使用构建器意味着你创建的库只包含你确实需要的元素。当然，其缺点是你必须提供对这个库的维护，这意味着当作者有更新时你得不到自动的更新。如果你选择了这一方案，关键是要定期检查是否有更新库的需要。

6.4.4　执行必需的测试

在将任何库放到生产环境中之前，你必须构建一个测试套件，以确定这个库是否能如你期望的那样工作。你可以使用一个供应商指定的测试套件，但使用自己设计的、能模拟你的应用程序如何使用这个库的测试套件也是很重要的。测试过程应该包含对合法和不合法的输入数据的检查。重要的是要确保对于合法的输入你能得到预期的输出，而对于不合法的输入你能得到某些异常。第 11 章和第 12 章详细探讨了如何对应用程序及其使用的库执行各种级别的测试。

6.5　区分库和框架

第 5 章的补充信息“区分框架和库”从可靠性的角度简要概述了库和框架的不同。当然，除了可靠性问题，它们还有更多的差异。重要的是要对如何准确区分两者的不同以及你为什么需要关心这些不同有更好的理解。

框架以库所不具备的方式为网站提供了一套模板。当使用库时，应用程序只是依靠其他来源的代码。框架的使用意味着另一种层面的参与，应用程序现在使用外部源来获得额外的支持。因为框架更充分地集成到一个应用程序中并且提供了更多的资源，所以比起使用库，你还必须将其放在一个更高的标准上（虽然你应该将两者都放在高标准上）。

尝试指出一个框架是否能安全使用是很困难的。事实上，测试会变得几乎不可能。为此，第三方的安全专家尝试量化框架和模板库的安全级别。查询相关信息的一个好地方是 mustache-security（https://code.google.com/archive/p/mustache-security/）。这个网站为你提供对框架 7 个级别的检查，如图 6-6 所示。

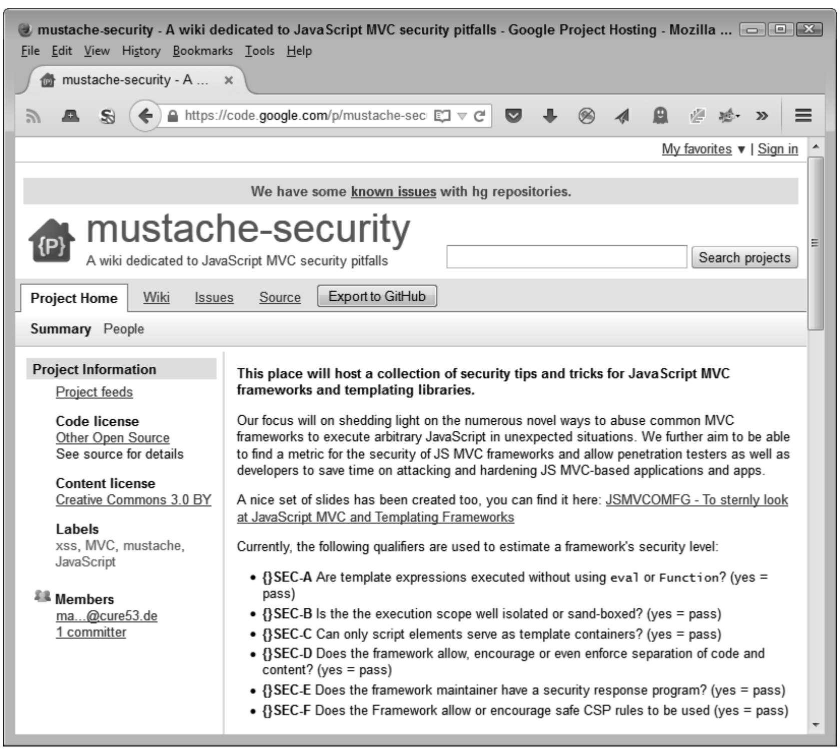
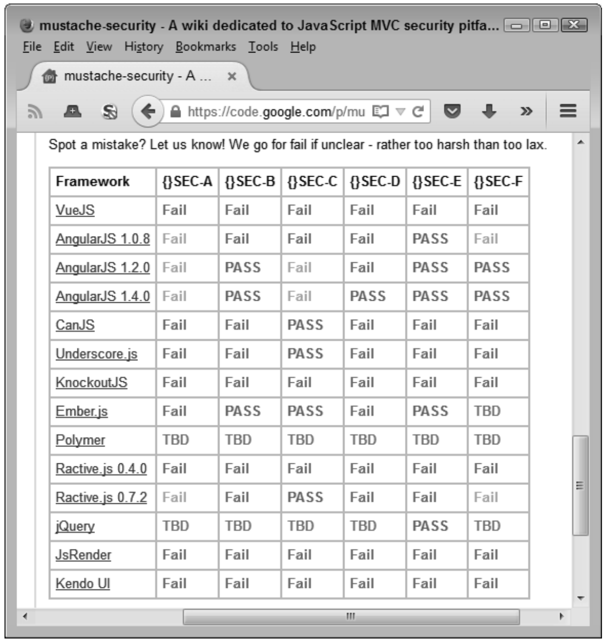


图 6-6：mustache-security 等网站提供了对框架 7 个级别的检查

看看这个页面的下面部分，你会发现与框架安全相关的新闻并不好。如今的大部分框架都包含严重的安全问题，你需要在将其应用于应用程序开发之前考虑这些问题。图 6-7 展示了某些如今最流行的框架和模板库的检查结果。



Framework	{}SEC-A	{}SEC-B	{}SEC-C	{}SEC-D	{}SEC-E	{}SEC-F
VueJS	Fail	Fail	Fail	Fail	Fail	Fail
AngularJS 1.0.8	Fail	Fail	Fail	Fail	PASS	Fail
AngularJS 1.2.0	Fail	PASS	Fail	Fail	PASS	PASS
AngularJS 1.4.0	Fail	PASS	Fail	PASS	PASS	PASS
CanJS	Fail	Fail	PASS	Fail	Fail	Fail
Underscore.js	Fail	Fail	PASS	Fail	Fail	Fail
KnockoutJS	Fail	Fail	Fail	Fail	Fail	Fail
Ember.js	Fail	PASS	PASS	Fail	PASS	TBD
Polymer	TBD	TBD	TBD	TBD	TBD	TBD
Ractive.js 0.4.0	Fail	Fail	Fail	Fail	Fail	Fail
Ractive.js 0.7.2	Fail	Fail	PASS	Fail	Fail	Fail
jQuery	TBD	TBD	TBD	TBD	PASS	TBD
JsRender	Fail	Fail	Fail	Fail	Fail	Fail
Kendo UI	Fail	Fail	Fail	Fail	Fail	Fail

图 6-7：大部分框架和模板库在安全方面还有很长的路要走

这类网站的有趣之处在于它们通常提供了这些框架如何失败的例子。例如，点击图 6-7 中表格中的 AngularJS 链接，你会看到该框架在安全领域失败的细节，如图 6-8 所示。这些信息还包含了可用于消除这一问题的解决方案，这也是你要花费一些时间来研究框架并确保准确理解它们的缺陷所在的主要原因。

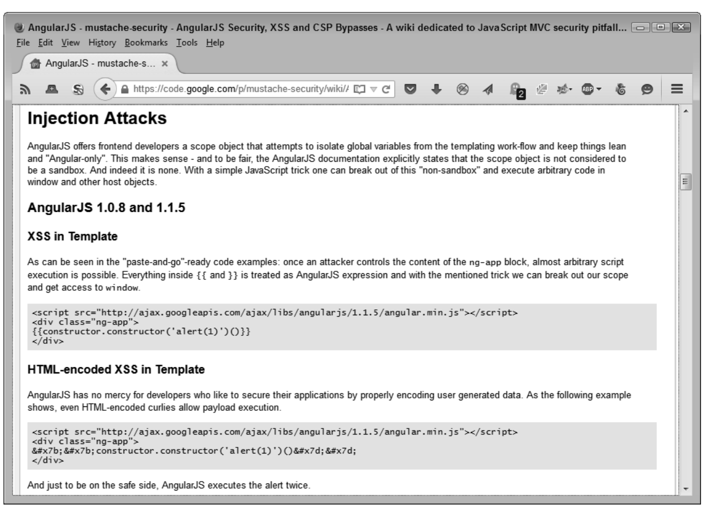


图 6-8：某些安全网址提供了示例代码来展示框架中的安全问题

第7章

慎用API

API 提供了一种使用别人服务器上成熟的应用程序来满足自己特定需求的方法。使用 API 意味着要将一个请求发送到一个黑箱并接收某种类型的响应。这样的请求就像函数调用一样简单，但通常需要你提供数据。响应可以是数据的形式，也可以是其他服务。例如，有些 API 用来执行开门这样的物理性任务。事实上，API 或许可以满足你的一般需求。本章将探讨使用 API 过程中的安全问题以及 API 与库的区别。

查看编码示例是发现 API 带来的安全威胁的一种方式。本章在解释过程中将使用一个简单的例子。虽然你可以为大量的应用程序任务使用目标 API，但本章的要点是创建一些用于讨论的应用任务，然后观察使用 API 会给应用程序带来什么问题。请将本章中的例子视为一个讨论点，而不是一种实际的编程技术。

> 虽然本章在编码示例和 API 出错的例子中探讨了某些 API，但其意图并不是要针对这些具体的 API。你所使用的每一个 API 都可能有严重的漏洞，从而使得黑客入侵你的系统或给应用程序造成其他问题。这些具体的例子会帮你理解所有 API 通常会如何出错。总之，你不应该将本章中的例子作为对具体 API 存在问题的暗示，进而避免使用它们。

当然，所有这些探索都是为了确定你应该如何安全地使用 API，将其作为应用程序开发策略的一部分。因为 API 有很多优点，所以你不会很快看到它们消失，尽管从安全的角度看，使用它们不是一个好主意。事实上，你可以期待 API 使用范围的扩大，因为开发人员一直有时间紧张（time-crunch）的问题，而 API 可以解决这一问题。随着更多的人使用基于 Web 的应用程序，且这些应用程序变得更加复杂，你可以期待 API 的使用会扩大以满足具体的需求。随着使用量的增加，黑客可利用的机会也会增加，他们想要窃取你的数据并以你从未设想的方式使用它。总之，API 的安全内存占用会增加，你在使用它们的过程

中要花费的努力也必须增加。

7.1 区分API和库

有些开发人员认为，除了自己的代码之外，所有其他的源都是一样的。但是 API 和库还是有明显的区别。例如，API 表现为一种进程外调用，因为它们在另一台服务器上运行，而库表现为进程内调用，因为你会将其作为应用程序的一部分包含在里面。进程内与进程外的明显不同会改变你看待安全问题的方式。接下来的几小节有助于更全面地定义 API 和库的区别，从而让你更好地理解使用这二者时的安全问题。

> 有些人将库和 API 作为可互换的东西使用，但在本书中，两者的意思有明确的不同。第 6 章详细定义了库，但从本质上说，库是你通过使用宿主语言所要求的任何技术包含到应用程序中的外部代码。你将库作为应用程序的一部分直接在自己的代码中使用。所以，尽管库的源代码存在于第三方服务器上，库通常也是一个进程内编码技术。为了使用库，要将第三方服务器上的代码下载到你的服务器上并成为应用程序的一部分，与应用程序在同一个进程中运行。
>
> 本书还会区分用于执行任务的 API 和作为应用程序开发工作一部分的 API。例如，网页 http://www.w3.org/standards/webdesign/script 上描述的 JavaScript Web API 定义了 JavaScript 的语言要素。这一类 API 定义了用于编写脚本的一门语言。是的，这是一种正规的 API，但不是本书上下文中所指的那种。本书的 API 更多的是指一种 Web 服务，而不是一种特定语言的规范。

7.1.1 考虑流行速度上的差异

库与 API 都能为开发人员节省开发时间。事实上，现在大部分基于 Web 的应用程序会结合使用库与 API 来执行任务。但是看起来 API 比库流行得更快，原因如下。

- *减少了资源的使用*
  API 在别人的服务器上运行，所以你不需要获取所需的资源。但是，代码在别人的服务器上运行也意味着你在某些节点上会失去对数据的控制，并且你永远看不到真正的 API 代码，这样也就不能对其表现的安全威胁进行评估。

- *减少了编码要求*
  通常而言，使用 API 比使用库更加简单，并且需要更少的代码。当然，你对 API 也会有更少的控制权。这是灵活性和复杂性之间的交换。由于开发人员越来越超负荷，可减少编码要求的 API 变得非常有吸引力。但是，被降低的灵活性也会成为潜在安全威胁的源头，因为你不再能修改代码以满足自己的具体需求。

- *更小的学习曲线*
  与库相比，学习如何使用 API 通常更容易，因为 API 通常是客户端 / 服务器模式的。最大的问题是如何格式化数据以让 API 明白你想要它做什么。最大的安全问题也就存在

于数据格式化当中。黑客可以制造格式不对的查询来导致服务器崩溃或迫使服务器以错误的形式响应。因为跟踪查询的源头可追查到你的应用程序，所以 API 所在的主机会惩罚你的应用程序，而不是惩罚造成问题的黑客。

7.1.2 区分用法上的差异

你不太可能在每个使用库的地方都换成使用 API（反之亦然）。某些情况需要一种集成的方法。例如，你不能简单地用 API 来编写 UI，编写 UI 真的需要库。（你可以获得用于 UI 的数据，如后面要介绍的 Google API 的例子，但这不同于真正创建界面元素。）接下来描述了 API 的一些典型应用以及它们的安全意涵。

- 数据查询

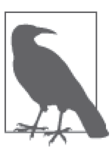
  API 最常用于请求一些数据。这样的请求通常包含请求数据参数。数据源可以是数据库、传感器、计算值和分析结果。只要你处理数据，中间人攻击就是很普遍的。但是，黑客可以简单地选择更改数据来误导你，或使用接收的数据作为入侵你网络的方法。

- 监视控制与数据采集

  物联网（The Internet of Things，IoT）依靠 Web 接口来执行很多监视控制与数据采集（supervisory control and data acquisition，SCADA）任务。例如，当你有一个合适的恒温调节器时，无论你在哪里，都可以通过智能手机上的 Web 应用程序轻松调节房子的温度。黑客可以轻松地接入到相同的接口并调低房子的温度直到水管被冻住，或使屋子热到你回到家时需要把门打开。当然，SCADA 涵盖包括工业控制在内的各种事情。在这种情况下，安全漏洞不仅会影响数据，还会造成物理影响。

- 监视

  数据查询与 SCADA 之间的中间地带是监视系统。例如，警察可能想要在抗议或其他事件发生时监控街道摄像头。通过 API 可以创建可在任何地点监控任何摄像头的应用程序，为广大群众提供保障。黑客可让这个 API 提供错误的信息。



你很容易会想到你或你的公司使用的应用程序并不真的适合作为一个监视系统。但是，房屋安全、婴儿监控、健康设备以及各种其他的应用程序都表现得像一个监视器。因此，监控软件领域可能比你最初想象的要大。

- 多媒体

  API 不仅仅处理文本或其他抽象数据。它还能执行与人的其他感官相关的任务，如视频和音频就是最普遍的。对多媒体的控制和操作创造出一种用户以非传统方式与应用程序交互的环境，且结果通常是令人惊讶的。通过将多媒体与数据科学结合，有可能让人们在日常生活中认识到新的模式和对象，比如紫外线的可视化场景，这在过去是不可能的。但是，很重要的是要认识到多媒体只是另一种形式的数据，而所有形式的数据都容易被黑客拦截、破坏，并且以未想到的方式被操作。

- 定位

  虽然地理定位是定位采集、查询和操作的最常见形式，但重要的是要理解存在处理各种

位置信息的 API。与多媒体类似，位置也是一种特殊的数据。在这种情况下，它提供了一个与某一具体目标空间相关的二维或三维的空间点，比如地球。就这一点来说，某些 API 还在公式里加入了四维元素：时间。想象一下，如果黑客选择修改两个对象，比如两辆汽车的位置信息，尝试让它们在同一时间占据相同的空间，会发生什么（导致相撞）。

7.2 用API扩展JavaScript

从现实世界的角度来考虑安全性是很重要的。接下来的几个小节描述了为了测试和应用开发目的而定位合适的 API 过程中的某些问题。然后我们会看一个用 Google Maps API 进行实践的例子。这个简单应用表明了我们需要考虑格式不正确的请求、服务器错误以及错误的返回数据等问题。这个例子很有趣，因为它展示了应该如何创建一个应用程序，根据 API 返回的数据用浏览器的画布画出一部分 UI。

研究本章示例的最好方式是使用可下载的代码，而不是手动键入这些代码。使用下载的代码可减少潜在的错误。你能在下载的代码的目录 \S4WD\Chapter07\GoogleAPI 下找到 Google Maps API 的例子。

7.2.1 定位合适的API

JavaScript 提供了一个同时支持 API 和库的可扩展编程环境。开发人员已经获得这两种形式的可扩展性的好处，通过扩展能创造出令人震惊的分享代码，可执行你可能想不到的任务。例如，可使用 JavaScript 应用程序与手机设备的振动功能交互以制造出特别的效果（可在网页 https://developer.mozilla.org/en-US/docs/Web/Guide/API/Vibration 上查看这一技术）。使用正确的资源，你可以找到执行你想要完成的任务的 API。下面列出了一些可用于查找你想要的 API 的位置。

- {API}Search（http://apis.io/）
- 40 个对 Web 设计师和开发者有用的 API（http://www.webdesignerdepot.com/2011/07/40-useful-apis-for-web-designers-and-developers/）
- API Directory（http://www.programmableweb.com/apis/directory）
- API For That（http://www.apiforthat.com/）
- APIs Dashboard（http://www.programmableweb.com/apis）
- API Data.gov（https://www.data.gov/developers/apis，点击“browse the current catalog for APIs”链接）
- Guide to Web API（https://developer.mozilla.org/en-US/docs/Web/Guide/API）
- MashApe（https://www.mashape.com/）
- Public API（https://www.publicapis.com/）
- Web API Interfaces（https://developer.mozilla.org/en-US/docs/Web/API）

上面只是在线 API 搜索网站的几个代表。它们当中的大部分都提供了对 API 的评论，某些网站还提供了检查。在选择使用某个具体的 API 之前，请尽可能多地阅读其相关的信息。

在测试系统里对 API 进行测试。确保使用抓包工具，对 API 在你的测试系统与主机之间实际传输的数据类型进行验证。以这样的方式来测试 API 听起来似乎有些偏执（请参阅 7.4 节，获取更多详细信息），但执行这样的测试有助于降低风险。记住，安全通常是风险和你在承受风险中获取的利益之间的平衡。

> 有些 API 提供商拥有非常多的 API，它们提供了自己的清单。例如，Google 提供了很多 API，而你会发现它们都列在了 GData API Directory（https://developers.google.com/gdata/docs/directory）中。重要的是要认识到很多专门的网站的清单包含有偏见的信息，所以通常建议你找其他网站查阅 API 的功能和缺陷。

7.2.2 创建简单示例

依靠 API 来创建安全应用程序比你想象的要更加困难，因为即使是最好的 API 也要依靠一定程度的信任。你的风险是要信任主机的数据或主机提供的其他资源。这一节用一个使用了 Google Maps API 的例子来看看安全问题是怎样产生的。为了使 API 可用，它还依赖 jQuery 和 jQuery UI（http://jqueryui.com/）库来展示数据及相关的 UI。你很少会在不依靠库的情况下使用 API 以某种方式与数据进行交互。

> 为了使用这个例子，你必须获得一个开发者密钥。Google 提供了两种密钥：付费的和免费的。运行这个例子你只需要免费密钥。付费密钥提供了更多的灵活性，而你可能在你创建的成熟应用程序中需要用到它。但是，就试验目的来说，免费密钥足以很好地工作。你可以在网页 https://developers.google.com/maps/licensing 上找到这一密钥。在开始使用 Google Maps API 之前，确保你完全理解了服务条款。在用 JavaScript 使用 Google Maps API 的过程中，你还能在网页 https://developers.google.com/maps/documentation/javascript/tutorial 上找到额外的帮助。

最好分几步来创建这个例子的代码。第一步是添加必需的 jQuery 引用，如下面的代码所示。

```
<script
   src="https://code.jquery.com/jquery-latest.js">
</script>
<script
   src="https://code.jquery.com/ui/1.9.2/jquery-ui.js">
</script>
<link
   rel="stylesheet"
   href="https://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
```

虽然这段代码引用了一个外部库，但库的代码是作为应用程序的一部分，以进程内元素的形式被包含进来的。第 6 章描述了关于库使用的各种安全问题。此外，你还需要添加对 Google Maps API 的引用，如下所示。

```
<script type="text/javascript"
   src="https://maps.googleapis.com/maps/api/js?key=Your Key Here&sensor=false">
</script>
```

这个例子完全不能生效，除非你将“Your Key Here”替换成你从 Google 获取到的密钥。因此，这一步很重要，因为这是你必须执行的一步，即使你正在使用从本书网站下载的代码。

库和 API 的引用都依赖 <script> 标签，但它们使用标签的方式是不一样的 。注意，库访问一个文件不需要额外的信息。而 API 要求某些额外的信息，在这个例子中包括一个密钥和一个传感器配置项。随着本节内容的继续，你还会发现其他的不同。

当你有了全部所需的引用之后，就轮到用画布来画地图了。画布就是一个简单的 <div>，如下所示。

```
</div>
   <div id="MapCanvas">
</div>
```

你必须提供给出 <div> 大小的样式信息，否则，即使 Google 发送给了你，地图也不会出现在屏幕上。这个例子使用了下面的样式信息。

```
#MapCanvas
{
   height: 90%;
   width:100%;
}
```

除了画布，这个例子还为输入提供了两个文本框，以及一个按钮，用来请求新的地图。这里界面并没有太多复杂的东西，但确实是可以工作的（这个例子预设好会在地图的中心展示威斯康星州密尔沃基的位置，你可以将其更换成你想要的其他地点的经度和纬度）。

```
<div id="Input">
   <label for="longitude">
      Longitude:
   </label>
   <input id="longitude"
          value="-87.95"
          type="text" />
   <label for="latitude">
      Latitude:
   </label>
   <input id="latitude"
          value="43.04"
          type="text" />
   <input id="submit"
          value="Get Map"
          type="button" />
</div>
```

这个例子的代码使用了许多 jQuery 和 jQuery UI 的功能，你已经在其他的应用中见到过。例如，这个应用程序创建了一个控制经度和纬度的选值框，以便更方便地逐步移动地图的中心。一次移动一个完整的度数对应用程序不会有太大用处，所以这两个下拉框每次只改变十分之一的度数（即使这样，这个度数可能还是太大了，而你可能想要改变它）。注意 `step` 选项用于执行这一任务。在接下来的代码中，`GetMap()` 函数是最重要的，因为它真正将地图展现在屏幕上。

```
$(function()
  {
     // 使用一个选值框来控制纬度
     var Latitude = $("#latitude").spinner(
        {
           min: -90,
           max: 90,
           step: .1,

           change: function(event, ui)
           {
              if (Latitude.spinner("value") < -90)
                 Latitude.spinner("value", -90);
              if (Latitude.spinner("value") > 90)
                 Latitude.spinner("value", 90);
           }
        });

     // 使用一个选值框来控制经度
     var Longitude = $("#longitude").spinner(
        {
           min: -180,
           max: 180,
           step: .1,

           change: function(event, ui)
           {
              if (Longitude.spinner("value") < -180)
                 Longitude.spinner("value", -180);
              if (Longitude.spinner("value") > 180)
                 Longitude.spinner("value", 180);
           }
        });

     // 这个函数会在屏幕上展示地图
     function GetMap()
     {
        // 构建发送给Google的参数列表
        var MapOptions =
        {
           center: new google.maps.LatLng(
                       Latitude.spinner("value"),
                       Longitude.spinner("value")),
           zoom: 8,
```

```
            mapTypeId: google.maps.MapTypeId.ROADMAP
         }

         // 向Google提供地图要在屏幕上展示的位置以及地图的参数项
         var map = new google.maps.Map(
            document.getElementById("MapCanvas"),
            MapOptions);
      };

      // 这个例子提供了两个获取地图的方法:在页面加载期间获取,或通过点击Get Map按钮获取
      $(window).load(
         function()
         {
            GetMap();
         });

      $("#submit").click(
         function()
         {
            GetMap();
      });
})
```

纬度的范围从北纬 90 度到南纬 90 度，所以这个例子反映了这一需求。类似地，经度的范围从西经 180 度到东经 180 度，这个经线位于英格兰的格林威治。你可以在网页 http://geography.about.com/cs/latitudelongitude/a/latlong.htm 上获取更多关于经纬度的信息。

GetMap() 函数执行了实际的获取地图的任务。要实现它，你的应用程序必须创建一份地图选项的列表。这个例子展示了一个简单但典型的列表。这些选项中最重要的就是地图的中心在哪里。在这个例子中，地图自动将它的中心置于威斯康星州的密尔沃基，但你可以将其更改为任何你想要的位置。这个例子使用了一个值为 8 的缩放因子，而你可以看到一幅街道地图。Google Maps API 实际提供了很多你可尝试的地图类型。

GetMap() 会在两处被调用。当应用程序加载时，你会看到威斯康星州的密尔沃基（除非你更改了默认设置）。在更改输入值之后，你还可以点击 Get Map 来展示新的位置。图 7-1 展示了这一应用的典型输出。

本章不会详细深入 Google Maps API，它相当复杂。本章会展示一些简单的例子，你能在稍后轻松地将其扩展到成熟的应用程序中。现在的许多公司会为了各种有趣的目的而使用地图。网页 http://blog.smartbear.com/software-quality/bid/242126/using-the-google-maps-api-to-addcool-stuff-to-your-applications 上的文章就如何在基于浏览器的应用程序中使用 Google Maps API 提供了一些额外的点子。

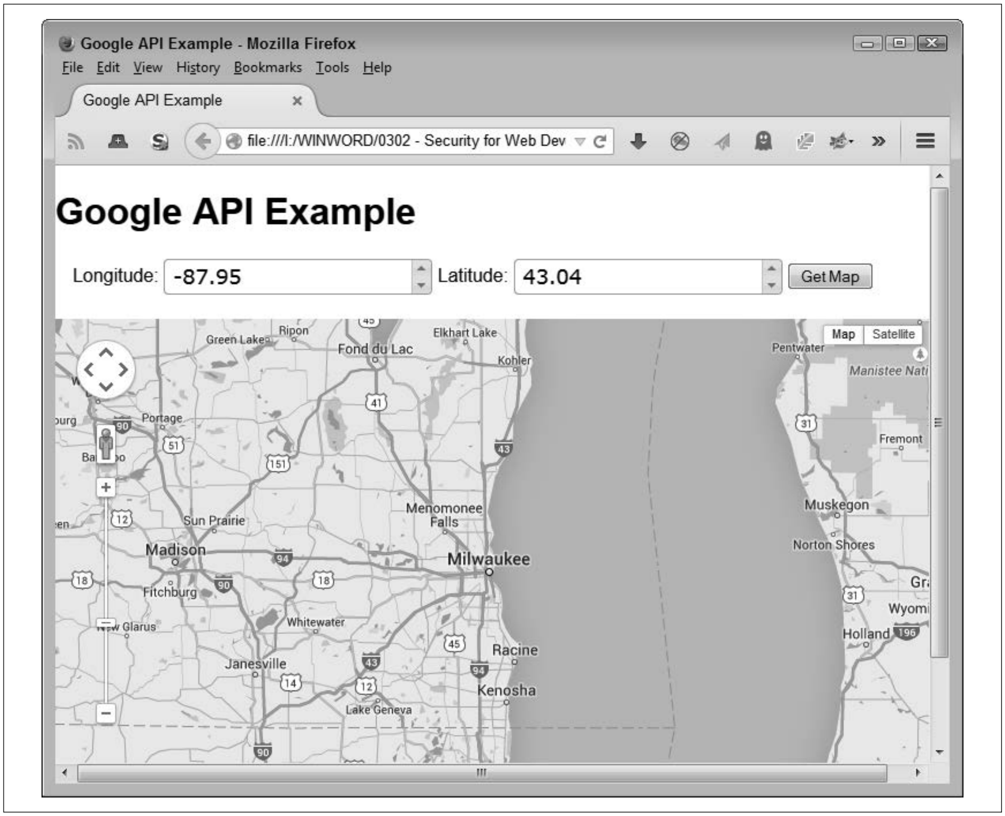


图 7-1：这个例子能展示任何经度和纬度的位置

7.3 定义API带来的安全威胁

作为一名开发人员，知道你可能面对什么类型的 API 威胁是很重要的。虽然接下来的几个小节可能没有包含最近的 API 威胁，但它们确实涵盖了各种各样的威胁，并且提供了可帮助你查找最新威胁所需的信息。这些真实的案例可能会让你理解你将遇到什么样的问题。

7.3.1 MailPoet毁了你的好声誉

很多人都会花大量时间关注博客。我会用我的博客（http://blog.johnmuellerbooks.com/）跟读者交流并告诉人们我自身最近的努力情况。你可以找到关于任何目的和任何主题的博客。人们将他们的声誉放在博客上，作为帮助世界成为一个整体的方式。所以，当博客的插件，比如 MailPoet（http://www.mailpoet.com/），出现问题时，会影响每个使用它的博客的声誉。准确地说，MailPoet 插件问题发生在 2014 年 7 月 1 日。

其缺陷允许攻击者通过远程上传将任何 PHP 文件放置到系统中（http://blog.sucuri.net/2014/07/remote-file-upload-vulnerability-on-mailpoet-wysija-newsletters.html），这意味着你的博客现在可能会用其过去经常提供的信息发起钓鱼攻击（http://wptavern.com/wordpress-mailpoet-plugin-

security-vulnerability-immediate-update-recommended)。最开始这种黑客行为会导致网站在某些情况下崩溃，但这个缺陷已经被修复，所以除非你很警惕，否则可能不会知道自己已经被黑。

这个缺陷的问题在于，发现问题的实体（Sucuri）在揭露这一问题之前没有给 MailPoet 足够的时间去修复并分发修复补丁（http://www.mailpoet.com/sucuri-hack-lessons-learned/），所以存在很多互相推卸责任的情况。这个插件已经被下载 17 000 000 次，但会被用在某些人的多个博客上，所以它有很大可能会导致问题。MailPoet 的威胁是一个在你开始开发之前需要验证 API 的安全性并且在继续使用时需要进行持续检查的例子。

在安全问题报告领域存在着很严重的问题。某些白帽黑客过度考虑自己的声誉，他们会报告重要的问题，却在发布信息前不给 API 提供商足够的时间去修复问题，从而导致其他黑客很喜欢的零日攻击。类似地，某些 API 提供商迟迟不着手修复能导致安全问题的 bug，这意味着黑帽黑客可以在 bug 被修复之前发现它们（再一次导致零日攻击）。进行披露的白帽黑客现在就能推卸责任，并告诉每个人他们已经让提供商意识到了问题。可能解决这一问题的一个方法是制定安全问题报告和修复的标准方法，这样每个人就能协同工作来保证软件更加安全。遗憾的是，在本书编写时甚至还没有讨论这一方案。

7.3.2　开发阅后即焚的图片

Snapchat（https://www.snapchat.com/）是一种很好的以短暂方式与别人交换图片的途径。这个服务的目标是让信息在一个较低风险的环境中自由流动。当每个人都遵守规则时它确实是有效的，但不是每一个人都能遵守［如 Snapchat 的安全课堂（http://blog.smartbear.com/apis/readyapi/security-lessons-courtesy-of-snapchat/）所解释的那样］。

由于第三方库的缺陷，API 很容易被黑。Snapchat 最近作出了回应（http://www.computerworld.com/article/2846038/snapchat-to-ask-users-to-stop-using-unauthorized-apps.html），它警告用户不要下载那些允许其破坏内在规则的第三方程序，而不是真的解决这一问题。在这种情况下没有提供稳定 API 的后果是 90 000 多张照片的泄露，这些照片可能会比任何知情的人透露的信息都要多。让我们讨论一下名誉的损失！当然，勒索金钱的事情会发生，并且不太可能会有人可以跟踪金钱方面的影响，因为没有人愿意谈论它。

这个 API 被黑的事件简而言之就是没有付费使用某些服务，无论它们看起来多么吸引人。这个威胁揭示了在你开始用 API 开发之前需要检查一下提供商的声誉。此外，它还表明你需要在使用 API 之前先研究一下它，因为很多黑客在行业杂志报道之前已经知道了这一缺陷。

7.3.3　使用“找回我的iPhone”却丢了手机

“找回我的 iPhone”（https://www.apple.com/icloud/find-my-iphone.html）可以在你弄丢 iPhone 时提供保护。你可以定位你的 iPhone，看看它在什么地方，并不让别人偷窥信息。但是，Apple 没有非常安全地访问 iCloud，这是提供“找回我的 iPhone”支持的服务。它很容易遭受暴力破解攻击，攻击者不停地尝试密码直到某一个密码成功为止。

有些黑客发现了这一漏洞并用它来锁定别人的 iPhone（http://ifwnewsletters.newsletters.

infoworld.com/t/9545692/122348447/801278/52/）。需要付出 100 美元，受害者才能使它们的 iPhone 解锁。如果受害者拒绝，黑客就会声称要清除所有数据。不花钱解决这一问题的唯一方法是将设备重新设为出厂设置。幸运的是，这个故事有一个开心的结局：黑客已经被逮捕了（http://www.computerworld.com/article/2490607/cybercrime-hacking/the-hackers-behind-those-iphone-ransom-attacks-have-been-arrested-in-russia.html）。没有人谈论有多少人支付了这笔费用，有多少人丢失了他们的数据，或有多少人的手机被黑了，但制造了这么一个大新闻，数量肯定不少。

这个威胁是一个需要检查输入和输出的案例。只要想想 API 是怎样工作的，开发人员就有可能很快找出暴力破解的攻击方法，进而避免潜在的问题。

7.3.4　Heartbleed泄露你最重要的信息

你依靠安全套接层（Secure Sockets Layer，SSL）在客户端与服务器之间提供安全的通信。不幸的是，与其他软件一样，SSL 会包含缺陷。在这个例子里，在 1.0.1 到 1.0.1f 版本的 OpenSSL（https://www.openssl.org/）中，使用一个称为 heartbeat 的特性，会允许攻击者获取用户密码和用于加密通信的长期私钥。

根据 InfoWorld（http://www.infoworld.com/article/2608736/data-security/heartbleed-to-blame-for-community-health-systems-breach.html）的报道，Heartbleed 应该为泄露 15 000 000 个病人记录的社区健康系统（http://www.chs.net/media-notice/）事故负责。这一事故造成的经济损失高达 1.5 亿美元（http://venturebeat.com/2014/08/25/community-health-systems-breach-could-cost-up-to-150-million/）。Heartbleed 的有趣之处在于它有很强的后劲。根据 ComputerWorld（http://cwonline.computerworld.com/t/8953013/1084055737/677451/17/）的报道，直到 2014 年 6 月 24 日，仍然有 300 000 台服务器受到 Heartbleed 的影响（离 4 月份发现这一缺陷有很长的时间来打补丁）。Heartbleed 缺陷如此严重，以致于它提供了自己的信息网站（http://heartbleed.com/）。

你应该将这一威胁看作警钟，不要再相信任何 API 以保证安全。黑客在持续地试水来看看他们能得到些什么。在这种情况下，对 API 的持续测试可能可以在缺陷变成问题前先获取其相关的信息。当然，仅仅依靠一种安全手段可能不是明智的做法。

7.3.5　遭受Shellshock攻击

某些专家宣传 Shellshock 比 Heartbleed 更糟糕（http://www.eweek.com/news/why-shellshock-bug-is-way-nastier-than-heartbleed.html）。当两者都非常糟糕且影响了大量的服务器时，很难说哪一个比另一个更坏。在这个案例中，用于 Linux 管理器的 bash 工具中的一个 bug 可通过创建格式不对的环境变量被黑，然后可运行一段 bash 脚本。对这一功能的访问可通过公共网关接口（Common Gateway Interface，CGI）或 PHP 提供。

这一黑客行为的最大限制因素在于其不是自动可被利用的（http://www.infoworld.com/article/2687975/security/four-no-bull-facts-to-know-about-the-shellshock-bash-bug.html），必需有访问服务器的外部接口。有些报告说，如 Yahoo!（http://www.bloomberg.com/asia）和 WinZIP（http://arstechnica.com/security/2014/10/white-hat-claims-yahoo-and-winzip-hacked-by-shellshock-exploiters/）这样的大公司已经通过 Shellshock 被黑了。潜在的 Shellshock 攻击很多，因为受

影响的服务器数量多于受 Heartbleed 影响的服务器。此外，一个可用的补丁要花很长时间才能开发出来（http://www.infoworld.com/article/2687750/security/attacks-against-shellshock-continue-as-updated-patches-hit-the-web.html），虽然现在已经有了一个（http://www.eweek.com/security/shellshock-vulnerability-finally-patched-as-exploits-emerge.html）。

这是一个要将数据严密保管的例子。通过提供关键数据的外部接口，公司会将自己置于数据丢失的风险中。保持敏感数据的严密是确保其安全的唯一方法。当然，你需要让某些数据可访问，但这应该是最后的选择且你应该只让必需的数据可被访问。

7.4　通过JavaScript安全访问API

如果你首先执行某种测试，是可以用 JavaScript 安全地访问 API 的。当然，测试会花费时间，并且你可能会在花时间确保 API 是正常工作的过程中遭遇阻碍。你唯一需要做的就是问问公司数据的价值与测试所花时间比起来哪个更值。答案通常是明显的，数据总是比测试时间更有价值。为此，接下来的几个小节描述了一些安全使用 API 的方法。

7.4.1　验证API的安全性

7.3 节描述了 API 带给开发人员的某些威胁。遗憾的是，行业媒体通常在你使用 API 实现了某一方案后才报道出这些威胁，令你很难在不消耗大量开支的情况下更换正在使用的 API。但是，通过查找其有关资料来验证当前的 API 是干净的，通常是一个好的开端。

提供商在安全方面的声誉也是值得一看的。新的提供商没有经过测试，因此不太可靠。比起在市场上已有产品且经过多次测试的提供商来说，使用新提供商会有更高的风险。在与经验丰富的提供商的对比中，一个事实部分抵消了新提供商的劣势，那就是黑客会瞄准市场份额较大的提供商，以增大获得真正有用东西的机会来作为他们付出努力的回报。

API 的构建方式、提供的文档以及调用所需的输入数量都反映了其提供的安全级别。为 Google 提供的某个服务进行注册是一件麻烦事，但你必须使用一个密钥来访问 API，这至少能减少一点 API 的攻击面。此外，Google API 不会让你提供敏感的数据；但黑客可能只通过查看你的请求就能发掘出某些数据。例如，请求某一经度和纬度的地图可能意味着，在这一地点有新的商业投资机会，或其他原因引起关注度的提升。

把 Google API 作为本章的例子是一个很好的选择，因为 Google 在安全性方面有很好的声誉。此外，在大部分情况下很容易验证你真的是在与 Google 进行交互，且你接收到的结果是正确的类型。Google 的文档可帮你执行各种检查来使得验证 API 的安全变得更加简单。

> 在检查 API 的安全性时，不要忘记要把通信的需求记在心间。很重要的是要确保提供商提供了 SSL 数据加密，以及通过使用双向 SSL 认证来确保双方都知道自己在跟谁对话。
>
> 但是，不要让安全的通信自身变成一种潜在的安全问题。黑客可以并且真的会偷取证书，这样他们就能提供所需的加密信息以验证一个安全的通信。你永远不会真正知道谁在网络的另一头。你所能做的就是提供必要的安全性，然后希望安全性足以强大到令大部分黑客无计可施。

7.4.2 测试输入和输出

一个简单的验证 API 可用性和安全性的方法就是简单地测试已知的输入输出。检查已知的输入和输出以确保 API 正常地工作。你可能会惊讶地发现某些 API 真的不能正常地工作。在某些情况下，这一问题可能并非是编码错误，而是没有理解 API 是如何工作的。

当然，很多公司会在应用程序的初始设计和开发阶段执行输入 / 输出测试。但是，创建一个测试框架可让你随机地执行测试。这一方法的好处是，这种测试可以帮你检测出潜在的数据损坏或其他潜在的安全问题。隔一段时间运行测试可帮你确保 API 是可用的。这些测试还能去除窃听者可能对你对 API 进行的查询做出的任何结论。

本章的 Google Maps API 的例子也包含了验证输入和输出数据的代码。为了让应用程序更容易理解，这些检查被去掉了。但是在现实的应用程序中，你需要在提交数据之前测试它们并且要验证你得到了你希望得到的返回值。当然，不可能去专门检查这一数据，因为地图在每一次调用时都会改变。



重要的是要确保随着时间的推移你使用的 API 是保持一致的。在某些情况下，提供商会发布一个 API 更新，引入有破坏性的更改。这一更改可能很小，以至于你一开始并没有注意到它，但黑客可能会在寻找潜在的安全漏洞时注意到它并加以利用。定期执行输入和输出测试可通过验证 API 在持续地正常工作，帮你消除这类问题，即使提供商执行了意料之外的更新。

7.4.3 保持数据的局部性和安全性

你发送给 API 的数据越多，别人截获数据的几率就越高。数据不需要以在途中被拦截的方式来展示问题。如果数据最终存在于主机服务器上，那任何访问这台服务器的人也可能访问到这些数据。保持秘密的最好方式就是不告诉任何人。尽可能少地与 API 分享数据是好的做法。

Google API 所使用的编码密钥是一个很好的安全措施，因为它肯定能辨别出你的公司。但是，密钥很重要还有其他的原因。在某些情况下，其他 API 会使用黑客可对公司进行追踪的识别码。任何拦截到 Google 密钥的人都不会了解到更多与你的公司相关的信息。

Amazon.com 是黑客可根据 API 要求的输入追踪到相关公司信息的一个例子。有些 API 过去常常依靠与公司相关的识别码，这些识别码还会被附在购买产品的链接上。黑客很容易发现这一识别码并能开始追踪公司的 API 调用。

7.4.4 防御性编码

当开发应用程序时，通常要假设输入和输出都是错误的，用户错误地处理各种事情，而且黑客正趴在你的肩膀上偷窥。是的，为了避免一些在应用程序中普遍存在的问题，开发人员需要培养一种真正的偏执狂意识。在阅读本书时，你看到了 API 沙盒等有用的技术可用于确定在与 API 交互时情况可糟糕到怎样的地步。重要的是不要对代码做任何假设，即使是你写的代码。

第 8 章

考虑使用微服务

微服务是一种相对较新的技术，它会将一个大应用程序分解成多个小模块。每个小模块独立运行并只执行单一任务。由于微服务所依靠的技术以及其运行的方式，微服务提供的安全性会比本书目前为止提到的其他技术更高。但是，与其他技术一样，微服务也会给黑客制造问题的机会。重要的是要记住任何技术都有让黑客可利用来做坏事的漏洞。开发人员的目标是要将这些漏洞减至最少，并确保有尽可能多的防护措施来帮助执行监控过程。

因为微服务很新，所以本章会花比平时略多的时间来介绍它。本书不会为你提供关于微服务的完整介绍，但你应该获取足够的信息来了解使用微服务的安全意涵，而不是固守你在过去使用的旧技术。此外，考虑人们在微服务场景中扮演的角色是很重要的。对微服务部署的敌视态度会导致一些安全问题，这些问题是你在开发阶段就需要考虑的。

本章会探讨你可能会如何创建自己的微服务（但不会给出源代码，因为本书是关于安全的，而不是关于开发微服务的）。这里的例子用 Node.js 和 Seneca 来创建一个简单的微服务，然后通过一个页面来访问微服务。这个例子主要是要讨论微服务如何工作，以便你更好地理解接下来要讨论的安全相关内容。结合使用 Node.js 和 Seneca 的原因是这些应用程序可以在 Mac、Windows 和 Linux 平台上运行。其他的微服务产品，比如 Docker，目前只能在 Linux 系统上运行。

本章最后会探讨拥有多个微服务访问途径的重要性。使用微服务的一个好处是你可以使用同一个微服务的多个副本来降低应用程序失败的风险。简言之，微服务会比单一应用程序更加安全和可靠。

研究本章所描述示例的最好方式是使用可下载的代码，而不是手动键入这些代码。使用下载的代码可减少潜在的错误。你能在下载的代码的 \S4WD\Chapter08 目录下找到本章的例子。

8.1 定义微服务

只能在单一平台上工作的应用程序最终会失去大部分用户。是的，它们会为特别的需求而继续存在，但大部分用户日常依赖的应用程序不会担心平台、编程语言要求或应用如今需要考虑的其他事情。微服务在当今的编程环境中可以工作得很好，因为它定义了一种新的看待代码的方式。不需要操心如何创建代码、将代码放到什么地方或采用什么语言，开发人员只需要考虑这段代码所执行的单一任务。这个任务可能在此刻并不一定适合应用程序，它可能只是展示了一些应用程序在处理数据时需要做的有趣事情。在应用程序开发的新时代，由于微服务等技术的出现，应用程序会在任何时间在任何地点运行。接下来会概述什么是微服务以及你为什么需要关心它们。

8.1.1 详述微服务的特点

许多开发人员习惯于严重依赖面向对象编程（object-oriented programming，OOP）技术来进行整体性设计。开发任何应用程序都始于定义各种对象和考虑各种问题。目前的应用程序设计需要大量的前期时间来启动，且它们被绑定到特定的平台上。微服务则不同。没有大量的代码，你只需要写很少的代码，并且在进一步的开发过程中才作很多决定，而不是在开始阶段就作决定。微服务具有以下特征。

- 小
  每个微服务只执行一个任务。
- 语言独立
  每个微服务使用最适合它要执行的任务的语言，而不会考虑其他微服务的需求。
- 数据传输独立
  虽然大部分微服务目前都依赖 JSON 来传输数据，但你可以使用任何最适合具体微服务的数据传输方式。
- 队列消息
  微服务的通信通常采用异步消息系统，所以没有哪个微服务会导致整个应用程序的延迟。
- 笨管道
  现今的许多通信方式存在的一个问题是管道智能化。微服务依靠笨管道以及智能服务。许多微服务采用 REST（Representational State Transfer，表述性状态转移）协议进行通信。
- 分散模式
  每个微服务与其他微服务是隔离的，并且与应用程序也是分开的。某个微服务的失败通常不会影响到应用程序的操作。每个微服务接受独立的监控。
- 与平台无关
  任何应用程序都能使用微服务，无论该应用程序及微服务在什么平台上运行。

你可能想知道微服务的大小，也就是只执行单一任务的真正含义。想想一个字符串的处理。当使用某个一体化应用程序时，会有一个单独的对象来处理首字母大写、反转以及将字符串转换成数字等操作。当使用微服务时，你会为每个任务创建一个单独的微服务。例

如，一个微服务执行字符串的首字母大写操作，而另一个负责字符串反转，还有一个将其转成数字。当你考虑微服务时，就要想到聚焦以及小规模。

从开发人员的角度来说，微服务代表着在灵活性和模块化方面的极致。你有可能在某一个时间只使用单一的一个函数而不干扰任何其他的配置。此外，因为微服务的更新很小，所以不会像 API 一样需要花费你大量的时间和精力。当出现错误时，修正也是一个小得多的问题。

8.1.2 区分微服务与库

认识到微服务并不像库那样以进程内方式运行是很重要的。微服务像 API 一样在某台服务器上运行。这意味着你在使用微服务时不会有使用库时碰到的安全风险。你可以完全将应用程序的代码与微服务分离开。

微服务的调用语法也与库不同，你要创建一个 JSON 请求并将其发送到服务器端。响应也会是 JSON 格式的。JSON 的使用让我们能够以不依赖 XML 的更丰富的方式进行工作。使用 JSON 比使用 XML 简单得多，因为 JSON 对于 JavaScript 是原生的并且有更轻量级的语法。你会在本章后面的部分看到它是如何工作的。而现在，你只需要知道在大部分情况下微服务不同于库的调用。

从安全的角度来说，微服务会比库更安全，因为它们不是以进程内的方式执行，并且你可以通过使用 JSON 的最佳实践方法来避免大部分错误数据输入形式。当然，黑客可以破坏你为安全所做的努力，而微服务也不会例外。

8.1.3 区分微服务与API

API 常常要求你创建一个对象，然后调用这个对象。请求可以有许多形式，如 REST、HTML 请求头或 XML。响应会涉及对页面对象的直接操作（如第 7 章展示的 Google Maps API 的例子）。这个过程是很麻烦的，因为你在使用大量不一致的一体化代码。

与 API 一样，微服务也是在进程外执行的。但是与 API 不同的是，微服务并不是大块的一体化代码。每个微服务都是很小的，并且在自己的进程内运行，这使得可以完全保证一个函数与另一个函数是隔离的。数据交换只使用一种形式，就是 JSON，这似乎是现在最好的方法，因为它比使用 XML 简单。

8.1.4 考虑微服务的策略

现在你知道微服务为开发人员、IT 以及整个公司都提供了很多好处。使用微服务是有意义的，因为它可以创建在任何地点任何设备上都能运行的应用程序，而不会给开发人员制造太多麻烦。不幸的是，一体化应用程序开发场景会创造一片封地，在这片封地里某一级别的管理者可以支配他们自己的特定资源。因为微服务很小，可方便地服务于各种目的，并且不会在意所需的数据来自哪里，所以它们打破了公司团队之间的围墙，破坏了过去被支配的各种封地。在这种情况下，某种程度的斗争，甚至是破坏的发生是在所难免的。

等式中被破坏的部分是你作为开发人员需要考虑的。不太可能有人会专门花时间去破坏一

个微服务项目，但在完成任务或正确做事方面的微小阻力会很容易破坏它。所有公司在面对新技术时都会有一种“我们之前从来没有这样做过”的态度，惯性在人们的每次努力中都扮演着一定的角色，所以在开始你的第一个项目时，必须找到克服惯性的方法。

如果你的公司成为了黑客的目标（或者有时候是通过偶然的机会发现的），那么从安全的角度来看，在项目早期阶段包含的漏洞会留给黑客彻底了解它并几乎肯定能加以利用的机会。为此，将某个微服务策略添加到你的编程工具箱中时，遵循一个流程是有帮助的（你不必精确地遵循下面列出的步骤，但它们会帮你克服在使用微服务时遇到的各种阻力）。

(1) 组建一支负责微服务开发的团队，让他们与目前维护一体化应用程序的团队分开。
(2) 从为一个新的应用功能创建一些粗粒度的微服务开始做起。
(3) 一开始开发提供自包含业务功能的微服务，这样你就不用太担心互动的问题。
(4) 给现有团队提供足够的时间去发现如何使用微服务并开始将它们用于现有应用程序中。但在你有足够的成功经验使每个人都同意之前，不要完全将现有应用程序迁移到微服务。
(5) 随着最初的微服务开发的进行，你会发现有哪些地方需要改变，创建细粒度的微服务会产生更好的效果。
(6) 标准化服务模板，这样就可以在最小化团队沟通成本的情况下开发微服务。标准化模板还可以减少安全问题，因为没有人必须去做任何假设。
(7) 创建足够细粒度的微服务去开发一个完整的应用程序，但不要聚焦于某个现有应用程序的需求，而要考虑创建一个新的应用程序。
(8) 获取必需的工具去执行粒度监控、日志收集、应用度量、自动化发布以及展示系统状态或日志报告等数据的状态仪表盘。
(9) 完全基于微服务开发技术构建一个小型应用程序。其目的是创建一个表明微服务是真的可行的完整应用程序。对于刚开始学习微服务的开发团队来说，开发一个小型应用程序会降低失败的概率。
(10) 慢慢地交叉训练每个人，使得技能方面的尖锐分歧减少。
(11) 打破不同团队之间的孤岛。开始创建微服务，使得来自每个团队的代码、资源和数据可提供给所有其他团队，而不需要考虑这些资源来自哪个团队。
(12) 从现有的一体化应用程序开发转移到充满微服务设计的开发。
(13) 从一个小型的一体化项目开始，如果可能，将这个一体化项目完整地迁移到某个微服务环境。慢慢地执行这项任务，在每次添加微服务之后都进行评估以确保应用程序真的运行得更快、更可靠并且更安全。
(14) 当你用细粒度和更多功能的微服务替换旧的微服务之后，将旧的微服务从系统中移除掉。

8.2 用JavaScript调用微服务

上一节帮你理解微服务是什么，但并没有向你展示微服务是如何工作的。接下来的几个小节会给出一个简单的例子，告诉你如何将微服务与应用程序放在一起并在应用程序中使用它。当然，你需要大量微服务来创建一个有完整功能的应用程序，但这个例子是一个很好的开始。

安装一个用于 JavaScript 的微服务

在开始使用微服务之前，你需要进行一次安装来支持它们。当然，你有很多方式可用于进行一次可用的安装，但有一种方式比其他都简单得多，那就是使用 Node.js 和 Seneca 的组合。

你首先需要在你的系统中安装一份 Node.js 的副本（如果之前没安装过）。安装所需的下载可从网页 https://nodejs.org/download/ 上获取。安装说明会对你有帮助。你可以在网页 http://blog.teamtreehouse.com/install-node-js-npm-mac 上找到 Mac 的安装说明，在网页 http://blog.teamtreehouse.com/install-node-js-npm-windows 上找到 Windows 的安装说明，以及在网页 http://blog.teamtreehouse.com/install-node-js-npm-linux 上找到 Linux 的安装说明。确保你使用的目录是允许开发人员访问的，例如，这意味着要在 Windows 系统中使用 C:\Program 文件目录。（如果可能，把产品安装到 Windows 系统的 C:\nodejs 目录下。）确保在安装之后运行建议的测试，以确保你的 Node.js 安装是正常的。

本书用 Seneca 来搭建微服务，因为它可以很好地在 Mac、Windows 和 Linux 系统上运行。当然，你可以使用任何满足你需求的 API 网关。只要安装了 Node.js，就可以根据网页 http://senecajs.org/install.html 上的说明，使用 Node 包管理器（Node Package Manager，NPM）来安装 Seneca。安装 Seneca 时，确保是在你计划用于创建代码的目录中，这样 Seneca 文件才会在正确的位置。可下载的源码包含了 Seneca 文件。

使用 NPM（https://www.npmjs.com/）使得安装各种 Node.js 的包都变得容易，且降低了创建各种应用程序的复杂性。很重要的是要注意到 Seneca 的安装说明展示了一个 `$` 提示符。当使用 Linux 之外的平台时，你会看到另一种提示符，但命令 `npm install Seneca` 在各个平台上都是相同的。很有趣的是，在安装的过程中提示符只会显示一个忙的提示，你不会看到任何说明某些事情正在发生的输出，直到最终看到 Seneca 的目录结构时。（如果你完全不能让标准 NPM 安装生效，那么通常可以直接从网页 https://github.com/rjrodger/seneca 上获取源码并以源码方式安装，但是手动安装肯定要困难得多。）

8.2.1　理解通信中REST的角色

微服务依赖 REST 这样的通信架构类型，因为它比其他协议［如简单对象访问协议（Simple Object Access Protocol，SOAP）］等更轻量。使用 SOAP 在某些条件下有优点，但在互联网场景中存在一些问题，比如占用大量带宽，以及在客户端和服务器之间需要一个更加正规的通信水平。依赖 REST 进行通信的应用程序被称为 RESTful 应用程序。在微服务中使用 REST 具有以下益处。

- 解耦消费者和生产者
- 提供无状态的通信
- 允许使用缓存
- 允许使用分层系统
- 提供统一接口

在微服务中使用 REST，你有大量选项可以选择。但是，最简单的方法（本例中使用的方法）是依赖某个特殊格式的 URL。例如，http://localhost:10101/act?say=hello 是本例中使用的 URL。在这个例子中，你用一个特殊的端口 `10101` 来联系本地主机。你用 `act` 发送了一条消息。这条消息会被解释为一个 JSON 名 / 值对，`{say:"hello"}`。这个例子展示了所有这些是如何工作的，但重点是你发送了一个请求，然后得到了一个 JSON 响应。是采用 REST 进行通信让事情变得简单。

8.2.2 用JSON传输数据

微服务依赖 JSON 来传输请求和响应。是的，你还会使用 REST 来发送数据，但这个信息最终是以 JSON 格式发送的。下面是你使用 JSON 来传输数据的三个主要理由。

- 干净的数据
  JSON 的数据格式是很直接的。数据表现为两种形式：名 / 值对或是一些值的列表。因为数据格式非常严格和简单，所以在传输数据时出错的机会较少，可靠性和安全性问题也更少。

- 效率
  因为 JSON 避免了 HTML 和 XML 中标签的出现，所以它会比其他类型的数据传输小。信息仍然以文本的形式进行传输，但格式本身是很高效的，这意味着你可用更少的资源来传输数据。

- 可扩展性
  JSON 所使用的严格数据格式意味着数据的传输是标准化的，这就可以根据需要更简单地扩展应用程序。使用单一的数据结构意味着你可以将你的代码放到任何你需要它的地方。

JSON 通常会使用 5 种不同的数据形式。不像 XML，你不会创建那种可遵循任何可想象的形式的复杂数据层级。下面是 5 种可用于传输数据的形式。

- 对象
  一个对象就是一个名 / 值对。名 / 值对出现在一对大括号（`{}`）中并且以一个分号（`:`）分隔。你可以用逗号来分隔多个名 / 值对以创建复杂的对象。例如，`{say:"hello"}` 就是一个名 / 值对。

- 数组
  一个数组包含着在中括号（`[]`）里的一个或多个值。例如，`["One", "Two", "Three"]` 是一个包含着三个字符串值的数组。

- 值
  一个值就是一个单一项。JSON 会将字符串、数字、对象、数组、`true`、`false` 和 `null` 作为值。

- 字符串
  在引号里的一系列字符（据说大部分的文本都使用双引号）。JSON 会识别前面带有反斜杠（`\`）的控制字符。这些字符是：空格（`\b`）、换页（`\f`）、换行（`\n`）、回车（`\r`）以

及水平制表符（\t）。你还能用 \u 和 4 位 16 进制数表示 Unicode 字符。例如，\u00BC 就是四分之一（¼）符。

- 数字

  数字是一些没有引号的数值字符，有或没有小数点。加号与减号表示正值与负值。你还可以通过添加 e 或 E 来用科学记数法表示数字。例如，-123e20 就是一个完全正常的值。

8.2.3　用Node.js和Seneca创建微服务

你可以找到很多用 Node.js 和 Seneca 创建在线微服务的例子。不幸的是，大部分都很复杂且难以使用。其中有一些过时了。最好的例子位于网站 http://senecajs.org/ 上。服务器的源码如展示的那样工作。但是，在 service.js 中可找到一个更简单的例子，如下所示。

```
require('seneca')()
   .add(
      { say:"hello"},
      function( message, done )
       {
         done( null, {message:'hello'} )
         })
   .listen()
```

在这个例子中，require('seneca') 将 Seneca 库加载到内存中。这段代码接着添加了一个 { say:"hello"} 的匹配模式作为 JSON 对象。与该匹配模式相关的 function() 会输出另一个 JSON 对象，{message:'hello'}。这个例子在创建 JSON 对象时专门使用了单引号和双引号来表明它们都是可行的，尽管官方的规范看起来并不是这样。最后一步是告诉服务 listen()。你可以往 listen() 函数中添加一个端口号。如果你没有提供端口号，这个服务会监听默认端口 10101。要启动这个服务，你要在命令行中输入 node server.js 并回车。你可以看到如图 8-1 所示的启动消息。

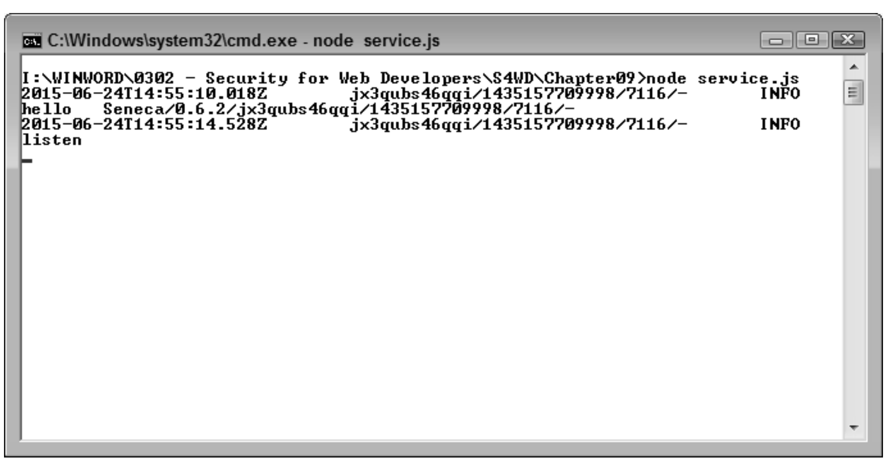


图 8-1：微服务正在监听请求

启动过程会记录两个步骤。首先是 Seneca 进程的初始化（就是在图 8-1 第三行里 Seneca 说了句“hello”的地方）。第二步是将微服务置于监听模式（如第五行所示）下。任何时候微服务发起了调用或执行其他任务（不是简单的输出），你都可以在窗口中看到一个或多个日志记录。从安全的角度来说，这可让你跟踪微服务的功能并检测是否有人尝试做一些你不希望发生的事情。

当然，你可能想要测试一下这个微服务。打开浏览器窗口并输入地址 http://localhost:10101/act?say=hello。微服务会输出一个简单的 JSON 对象，如图 8-2 所示。

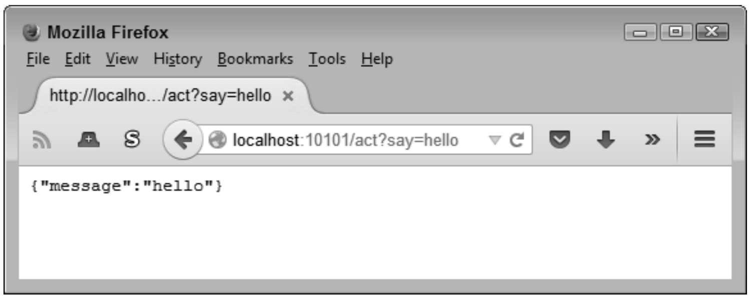


图 8-2：这个简单的例子会输出一个 JSON 对象

当你回去看控制台窗口时，不会看到任何东西。那是因为这个函数输出了一个简单的 JSON 对象而没有做任何环境外的调用。但是，请尝试输入 http://localhost:10101/act?say=goodbye 这样的请求。现在你可以看到控制台窗口中有一些东西了，如图 8-3 所示。

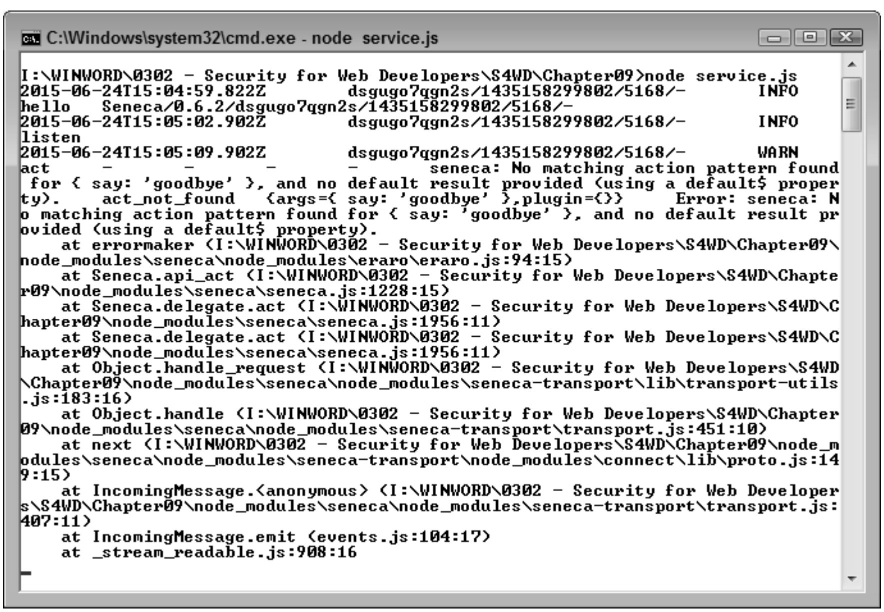


图 8-3：输出文本中有大量错误信息

输出中包含一个调用栈，在这个例子中你可以忽略它，但在使用复杂的微服务时它很有用。在这个例子中最重要的信息出现在顶部。你会看到一个警告说没有与`{ say: 'goodbye' }`匹配的模式。注意 REST 请求被转换成一个 JSON 对象。错误输出会更精确地告诉你发生了什么，所以发起非法的请求会变得更加困难。

这真是一个做试验的绝佳例子，因为你可以看到尝试愚弄微服务的 REST 通信的结果，而不用担心其他因素会影响这一结果。当你使用完这个例子之后，按下 Ctrl+C 组合键或 Ctrl+Break 组合键来停止服务。该服务会停止，而你会看到命令提示符再次出现。

8.3 定义微服务带来的安全威胁

当谈到安全问题时，微服务在许多方面与 API 类似。例如，微服务可能会遭受中间人攻击。黑客真正从这种攻击中获得的好处会比 API 少，因为微服务是小型的、自包含的，并且只执行一个任务。但是威胁仍然存在，并且黑客真的只需要一种攻击手段就能毁灭你美好的一天。虽然你可能会读到很多文章告诉你微服务会提供天然的安全性，但它们仍然有安全问题，并且你也需要了解这些安全问题。接下来的几小节会给出在处理微服务的安全性时需要考虑的优点和问题。

8.3.1 缺少一致性

微服务可能带来的最大问题是缺少一致性，这似乎也困扰着过去创建的每个库和 API。库以及 API 的开发人员一开始会想创建易于使用且一致的接口，但随着时间的推移，库或 API 就变成了一堆冲突策略的混杂物，这让难题变得相对容易解决。试图修复任何一种代码基础都是相当困难的，因为开发人员将库或 API 作为单一的一块代码使用。这些不一致性会导致安全问题，因为使用这些代码的开发人员以为这些调用会以某种方式工作但实际却不是。其结果就是在代码和使用它们的应用程序之间会有不匹配的情况出现。黑客会将这种错误作为入侵应用程序、数据或其运行所在的系统的一种方法。

微服务也会遭受缺少一致性的问题。关键是你要在开发过程中尽可能早地创建一个精确描述如何调用微服务的模板。与库和 API 一样，黑客会利用不一致性作为攻克你的安全措施的方法。与库和 API 不同的是，不一致性只会影响一个微服务，而不是整套代码。修复这个微服务会比较简单，因为你只需要查看一个调用，而不是整个 API。发布一个新的微服务也会比发布一个新的库或 API 更简单。因此，克服一个微服务的不一致性是相对简单的。

8.3.2 考虑虚拟机的角色

所有微服务通常都在自己的虚拟机（virtual machine，VM）环境中运行。这意味着一个微服务通常不会破坏另一个微服务。即使一名黑客入侵了一个微服务，他能造成的破坏通常是最小化的。但是，很有可能在同一个虚拟机中运行着多个微服务，为此，黑客可能尝试各种技术来入侵整个 API。为了最大化安全性，你要避免像这里的 service2.js 例子中那样堆叠微服务（要访问这个例子，你必须使用 999 端口，比如 http://localhost:999/act?say=goodbye）。

```
require('seneca')()
   .add(
      { say:"hello"},
      function( message, done )
       {
         done( null, {message:'hello'} )
         })
   .add(
      { say:"goodbye"},
      function( message, done )
       {
         done( null, {message:'goodbye'} )
         })
   .listen(999)
```

最佳实践是让每个服务在隔离的虚拟机上运行，以确保每个服务有自己的地址空间和进程。分离每个服务能使出现错误和黑客利用代码缺陷造成问题的机会变少。

8.3.3 使用JSON进行数据传输

没有完美的数据传输方法。是的，JSON 是轻量的、易于使用的，并且与 XML 等其他技术相比，不易遭受安全问题。但是，黑客仍然有很多方法来给 JSON 造成问题。接下来会描述更多普遍存在的问题。

1. 考虑eval()的危险

某些人有可能会发送一个响应给客户端，这个响应里面包含一个 `<script>` 标签并有一段可执行任何操作的脚本（在某些情况下，你甚至不需要标签，而只需要传递脚本）。幸运的是，现在的大部分浏览器会检测这种尝试并拒绝传输这种信息。例如，试试 http://localhost:999/act?<script>alert('Hello');</script>=goodbye，你可能会看到如图 8-4 所示的输出。此外，微服务本身会拒绝处理这样的请求。但是，这个脚本过于简单，而一个更加高级的尝试可能就会取得成功。

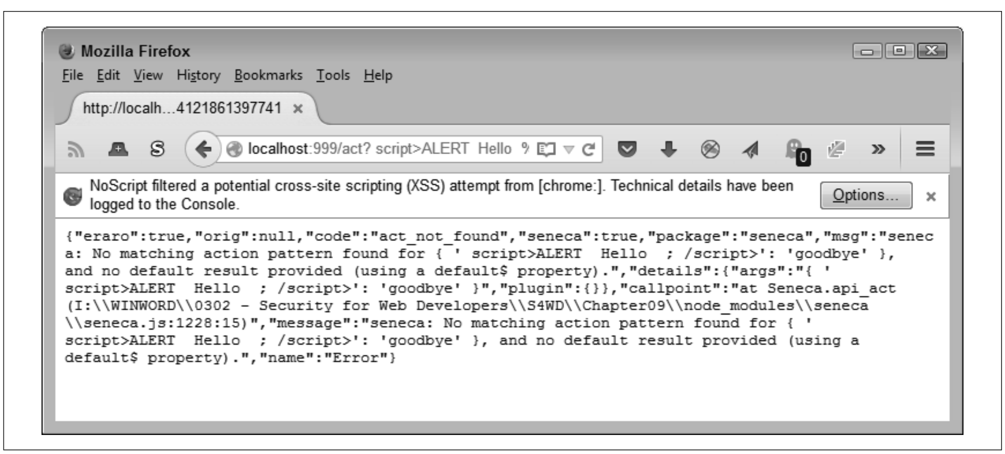


图 8-4：像 API 一样，脚本问题也会给微服务带来麻烦

2. 防御跨站请求伪造

一个跨站请求伪造（cross-site request forgery，CSRF 或 XSRF）是攻击者尝试让用户在不经意间或不知情的情况下执行代码的攻击。这个代码在用户所在的权限级别上运行并且有用户的认证凭据，所以它看起来是用户本人在执行代码而不是别人。在大部分情况下，当发生下面的一系列事件时，微服务会遭遇到这种特殊的攻击。

(1) 一名用户登录进一个使用微服务的应用程序。
(2) 该用户执行各种任务，每个任务都依赖 REST 进行通信。
(3) 攻击者发送给该用户一个有特殊格式的 URL，它看起来像是其他的 REST 消息，但会干黑客想要用户干的事情。这个 URL 可能作为电子邮件消息的一部分或在某些其他的通信中出现，甚至可能是网页里的一个链接。
(4) 用户初始化请求，就像平时一样发送 URL 到微服务。
(5) 恶意的请求被执行，而用户甚至可能还没意识到其已经发生了。

> 一个现实的攻击例子是 2008 年的 μTorrent 事故。这种攻击对于微服务开发人员很重要，原因是它发生于有 Web 服务器在后台运行的系统中。当攻击发生时，它会损害整个系统，这就是为什么你要将这些微服务放在自己的虚拟机中相互隔离开来。你可以在网页 http://xssniper.com/blog/2008/04/21/csrf-pwns-your-box/ 上读到相关的资料。

由于这种攻击工作的方式，你真正需要做的是确保应用程序在用户保持不活跃状态一段时间后自动使其退出。此外，你需要查找怪异的使用模式并限制用户在未经批准时所能做的事情。例如，用户能在银行转 1000 美元，但转 10 000 美元则需要管理者的批准。使用多步骤的工作流，这样攻击者需要与用户有更多交互才能使攻击奏效，这在某些时候也能阻止这类攻击，但用户会对必须执行更多步骤才能完成给定任务不满。

8.3.4 定义传输层的安全

微服务和 API 的致命弱点是需要来回发送消息，而不是在单一机器上执行任务。解决这一问题的基石在于传输层安全（Transport Layer Security，TLS）。关键是要确保客户端与服务器之间的传输层在消息发送过程中是安全的。这意味着要用 HTTPS 和 REST 等技术来确保通信尽可能安全。

把每个调用和响应都包在 HTTPS 和 REST 中的一个问题是应用程序会变慢。解决这一问题的最佳方法是使用负载均衡来传输客户端的通信，并保持一个渠道开放给后台处理需求。保持放开后台处理渠道可降低成本并减小使用 HTTPS 和 REST 的影响。

> 在公开网络使用 HTTPS 的一个问题是你必须从认证机构（Certificate Authority，CA）获取一个证书，这是一项很昂贵的支出，可能会使某些公司放弃使用 HTTPS。当你控制了通信渠道的两端时，就可以创建自己的证书，以大大降低的成本来达到相同的目的。

HTTPS 和双向 TLS 的使用可确保客户端和服务器在每次的请求 / 响应中都能识别对方。认

证减少了某些人能成功实施中间人攻击来非法获取数据的机会。现在的大部分通信是采用单向 TLS 的，也就是客户端会验证服务器的身份，但服务器简单地认为客户端没有受到损害。鉴于微服务的实质，你真的需要实现双向 TLS 来验证客户端和服务器的身份。

8.4 创建可替换的微服务路径

许多开发人员不太理解的一个东西是微服务的去中心化性质。每个微服务都是分开的。你不必考虑微服务需要的平台、它所使用的语言或其运行的地方。有可能有两个微服务，它们是用不同的语言开发的，在两个不同的地方在两个不同的平台上运行，却执行着相同的任务。由于这两个微服务所使用的环境非常不同，所以影响一个微服务的问题不太可能会影响另一个。因此，保持着两套微服务是一个好的做法，这样你就能根据需要切换它们来让应用程序持续运行。这也是这一节所要谈及的，即考虑用多个途径访问多个微服务的意义。

当考虑多个微服务以及它们所采用的途径时，也要考虑一下端口这样的事情。你可以创建在不同端口上运行的微服务。通常，你会使用在端口 999 上的微服务来执行某个应用程序所要求的任务。但是，如果端口 999 上的微服务变得过载、被损坏或者不能工作了，你可以切换到不同端口上的相同微服务。你的代码可以保持不变，而只是端口变了。采用这一方法会让你的应用程序有弹性、可靠并且灵活。这也意味着如果发生像分布式拒绝服务（DDOS）这样的攻击，你会有一些可选的方案。

第三部分

创建有用及高效的测试策略

你不太可能避免应用程序潜在的安全问题，但可以通过花时间思考和运用适当的测试技巧来减少问题。这一部分将探讨如何降低应用程序被攻破的风险。第 9 章将帮你像黑客一样思考，这是很重要的练习，将使你真正看到应用程序的安全漏洞。别人不会在你的安全漏洞上用大大的红字标上“来修复我！”，所以这个思考的过程相当重要。

第 10 章将讨论沙盒技术。把代码放在沙盒中不会真的让它没有问题，但可以减少代码可能带来的破坏。沙盒会让你的系统及其资源很难被访问，并且在涉及安全问题时，它会使一切都变得不同。

第 11 章和第 12 章将讨论各种测试。第 11 章关注内部测试，这通常有助于定位主要问题，但可能不足以找出以后可能会让你付出代价的细小问题。第 12 章讨论第三方测试，这对于缺少安全专家的小企业来说是非常重要的选择。无论你在哪里执行测试，对于了解在某一产品上使用应用程序的风险来说，进行充分的测试都很必要。

第 9 章

像黑客一样思考

大部分开发人员把时间花在考虑事情应该如何运作上面（即关注在代码正确的情况下事情会如何运作）。思考事情无法运转（即尝试确定代码出错时事情出错的方式）的想法是有些异类的。是的，无论是怎样的想法，开发人员都要时刻处理 bug，但他们的思路是不一样的。当你像黑客一样去思考时，可能在使用表面上完全可接受的代码，它可能没有 bug，但可能有安全漏洞。

本章包含一个帮你以黑客的视角看待代码的过程。你可以使用工具来查找潜在的安全漏洞，创建测试系统来尝试破坏代码，并依靠一些常见漏洞来使你的生活更轻松一点。黑客喜欢 BYOD 现象，因为这会让所有这些不安全的系统飘荡在各处，而 IT 人员对这些操作系统并没有太多经验。当然，总会有最终的应用程序测试人员，那就是用户。用户可以找到比任何开发人员能考虑到的更多的破坏应用程序的方式，但用户测试的价值在于可以发现你所做的假设真的是无效的。

事实上，你需要沿着这些思路去思考，这会推动很多公司雇用安全专家来考虑黑客会采用的各种迂回方式。黑客用这些方式来彻底破坏功能性应用程序，努力获得他们能有效利用的机会去执行特定的任务。但是本章会假设你的公司现在确实没有任何安全专家。你可以在应用程序接近完工时才考虑这个问题，但那时候想要低成本地修复问题通常太晚了。在审视应用程序时像黑客一样思考可以为你的公司节省金钱、时间和精力，以及最重要的，避免难堪。

9.1　定义Web安全扫描的需求

Web 安全扫描的基本思想是，它们会告诉你你的网站目前是否干净，并且有时候能帮你考虑潜在的安全漏洞。如果你有一个很大的企业，那么购买一个 Web 安全扫描器是个不错的

主意。但是，小企业常常可以使用免费的在线 Web 安全扫描器。当然，这些产品实际上不是完全免费的。当 Web 安全扫描器找到了某个系统问题时，其所属的公司通常会让你购买它来清理系统并提供后续保护。（在计算机的世界里真的没有免费的午餐。）

> 任何 Web 安全扫描器都做不到 100% 准确地检查出网站的潜在问题。对于远程扫描器来说更是如此，它们并没有在能够访问你想要测试的服务器的系统上运行。扫描器可以很好地定位现有问题，但你无法找到所有问题。
>
> 你还必须认识到 Web 安全扫描器也会误报。这意味着扫描器有可能会告诉你一个实际不存在的安全问题。
>
> 解决遗漏问题或误报的最佳方法是不止使用一个 Web 安全扫描器。依靠单一的 Web 安全扫描器确实是一个好的开端，但你需要付出超过实际所需的努力，才能构建一个安全漏洞最少且完全可工作的网站。

在查找更多关于 Web 安全扫描器的信息之前，很重要的是要弄清它执行什么任务。这里的例子会使用 Securi（https://sitecheck.sucuri.net/），这是一个能检查很多安全问题的免费扫描器，主要检查潜在的病毒感染风险。图 9-1 所示的主页会让你输入网站的 URL 并点击“Scan Website!”来开始扫描过程。

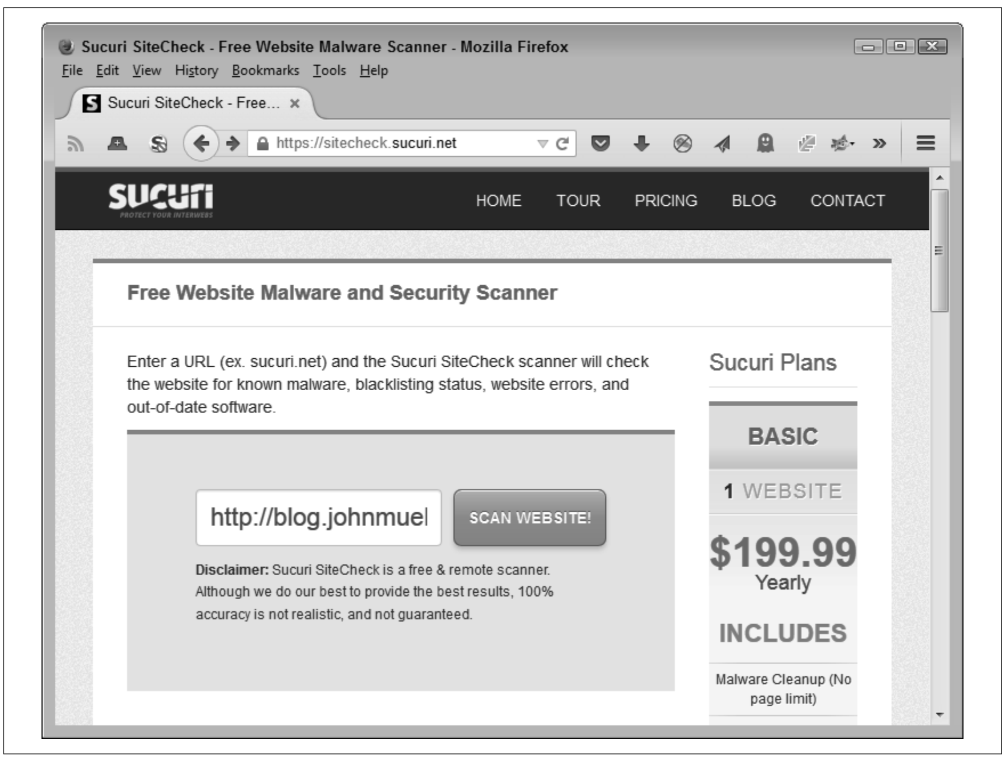


图 9-1：要开始扫描，需要输入你的网站 URL 并点击按钮

如果你有一个网站，请一定要查看提供主机服务的供应商。许多供应商会提供低成本的 Web 安全扫描器。例如，你可以在网页 https://www.godaddy.com/security/malwarescanner.aspx 上找到提供给 GoDaddy 用户的扫描器。供应商在确保你的网站不会遭受危害方面有既定利益，所以提供这种低价服务对他们也有好处。

在扫描完成之后，你可以看到扫描结果的概览以及一个更加详细的检查列表，如图 9-2 所示。当然，如果你想要一个更详细的检查，可以购买一份在网站右边列出的 Securi 计划。但是如果你只想看看扫描器能做什么，使用免费版本就足够了。

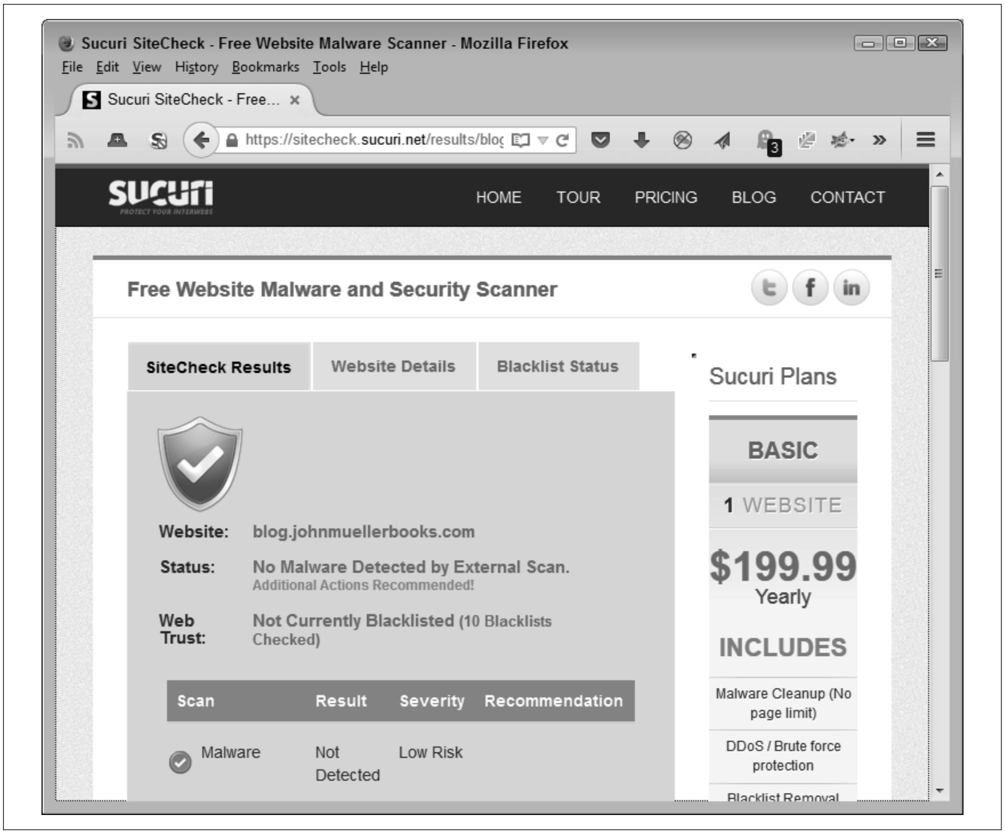


图 9-2：扫描完成之后会显示你的网站的各种问题

查看网站的 Website Details 选项卡下面的信息是很有趣的。图 9-3 展示了我的网站的信息。很有趣的是，黑客可以利用类似这样的扫描器来寻找你的网站中他们可能感兴趣的地方，比如所用软件的版本。Web 安全扫描器会帮你理解黑客所能掌握的各种信息。看到这些每个人都能找到的信息通常会有些可怕。

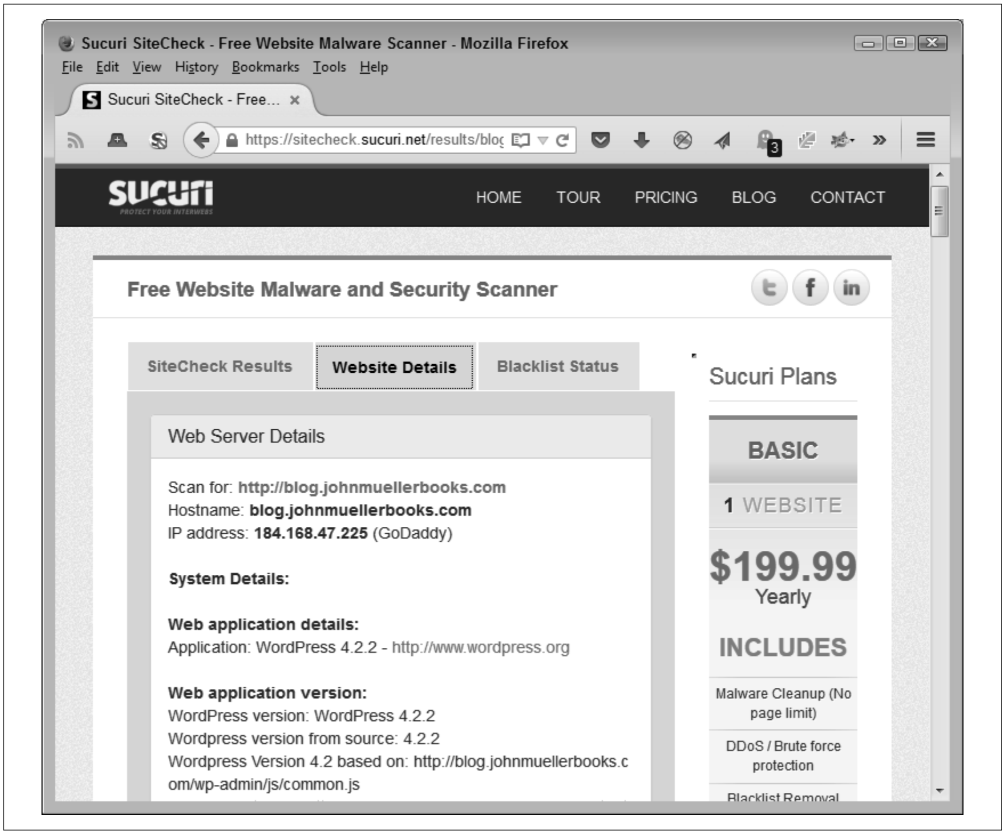


图 9-3：这些详细信息可以用来洞察黑客所能掌握的关于你的网站的信息

从另一个角度来看，详细信息也是有帮助的。当你使用一个主机网站时，就像许多小公司所做的那样，很难知道提供主机服务的供应商是否真的不断更新软件。Securi（及其他）Web 安全扫描器会认真检测网站并输出版本信息的列表，这些信息在你使用自己网站的仪表盘时可能从没看到过。例如，一个库可能还在很好地工作，但已经过时并可能会对系统造成安全漏洞。打电话给提供主机的公司通常可以快速解决问题，但在打电话之前你需要知道是什么问题。

你可以在网上找到 Web 安全扫描器的清单。其中比较好的一份清单是 Web 安全应用协会（Web Application Security Consortium）的应用安全扫描器列表（http://projects.webappsec.org/w/page/13246988/Web）。这些作者已经将这份清单分成商业、软件即服务（Software-As-A-Service，SAAS）以及免费开源工具三个类别。另一份好的工具清单位于开放 Web 应用安全项目（Open Web Application Security Project，OWASP）的网站（https://www.owasp.org/index.php/Phoenix/Tools）上，你可以在该网站上找到各种工具。

9.2 构建测试系统

当你像黑客一样思考时，有一个不必担心其被破坏的系统通常是很好的。事实上，系统很可能在中间某个环节就被破坏了，所以构建一个能被轻易恢复的系统是很重要的。接下来会概述构建测试系统时需要考虑的事情。

9.2.1 考虑测试系统的使用

你需要一个测试系统，这样你就能自由探索黑客会采用的各种攻击手段。除非充分了解如今的大部分应用程序是何等脆弱，否则你真的无法修复自己的应用程序。开发人员会犯各种错误，在当时看起来是无害的，但后面会导致安全问题。小到代码中的一些错误注释都可能导致问题。使用生产系统来测试是一个糟糕的想法，因为本书所描述的方法可能会让你遭受真正的攻击。你需要一个能够破坏并恢复的系统，以确保你真正掌握所学。

这里的要点是对有害条件下的各种应用程序进行测试，但是以安全的方式进行，不会真的损害任何真实数据或任何生产系统。给测试环境注入病毒可以让你发现应用程序会如何应对，而无需真的在生产环境中试验这个病毒。当然，你的测试系统不能连接到你的生产网络（否则病毒可能会跑出来并感染生产系统）。你也可以使用这个环境来测试各种漏洞，并确定黑客会采用什么手段来破坏你的应用程序。

但是，测试系统不止能用来测试应用程序。你还可以用它测试对策的效率或者特定配置的漏洞。测试整体的系统有助于确保应用程序不会轻易被其他部分的漏洞所破坏。测试整个环境是很关键的，因为任何黑客都会这样做。

9.2.2 接受必需的训练

在开发人员真正进行测试之前，要明白要测试什么以及如何测试。你可以通过使用 OWASP Security Shepherd（https://www.owasp.org/index.php/OWASP_Security_Shepherd）这个项目来开始训练过程，这个项目会给出应用程序的十大安全风险。它可以提供针对单个人的教程，你也可以在教室里用它给多名学生授课。这个应用程序提供了一个竞争环境，所以从某些方面来说这是一场安全游戏，应用程序会记录每个会话的分数。这个应用程序支持 60 多个级别的训练，你可以根据用户的学习曲线和经验来配置它。

在学习了基础知识之后，就需要花一些时间像黑客一样真的入侵软件，这样就可以了解到安全环境的另一端。当然，你要在安全和合法的环境下执行这一任务，这样才不会破坏自己的软件。OWASP 专门设计了一个叫 WebGoat（https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project）的应用程序来提供一些漏洞，应用开发人员可用它来发现在现实的应用程序场景中入侵是如何发生的。网页 http://webappsecmovies.sourceforge.net/webgoat/ 上的影片会带你一步步了解黑客做事的过程，这样你能更好地理解黑客正在对你的应用程序做什么。训练包括关于以下内容的影片。

- 通用原则（General principles）
- 代码质量（Code quality）
- 并发（Concurrency）

- 未验证的参数（Unvalidated parameters）
- 访问控制缺陷（Access control flaws）
- 认证缺陷（Authentication flaws）
- 会话管理缺陷（Session management flaws）
- 跨站脚本攻击（Cross-site scripting，XSS）
- 缓存溢出（Buffer overflows）
- 注入缺陷（Injection flaws）
- 不安全的存储（Insecure storage）
- 拒绝服务（Denial of service，DOS）
- 配置（Configuration）
- Web 服务（Web services）
- AJAX 安全性（AJAX security）

最后一步是要接受一个 WebGoat 的挑战来展示你新学到的技能。你要攻破认证措施，窃取所有的信用卡，然后使网站颜面尽失。其要点是从黑客的角度来完全理解黑客行为，但是，是在一个安全（并且合法）的环境中。

很重要的是要理解如何完善你的安全测试技能。网页 http://blogs.atlassian.com/2012/01/13-steps-to-learn-perfect-security-testing-in-yourorg/ 上的文章会教你 13 个步骤，你可以遵循这些步骤来改进自己的测试方法。将这些信息与这一部分介绍的工具相结合，可以帮你测试任何应用程序，即使你的公司缺少进行深度测试的资源。

9.2.3 创建正确的环境

你构建的任何环境都必须与实际的应用环境匹配。这意味着要能访问你计划支持的每一种操作系统、用户计划使用的各种浏览器以及每个系统会使用的对策（在某些情况下可能根本没有）。请确保每个系统都能轻易访问，因为你可能要重新配置它们来模拟用户使用的各种系统。此外，你必须定期计划重新构建测试系统。

> 因为你会用测试系统来进行各种测试，所以必须为你的实验室提供物理上的安全性。否则一些好心人会让病毒或恶意代码进入到生产环境中。你也不应该把安全设置开放给那些心有不满的职员，或那些会用它来给你的生活制造麻烦的人。测试环境应该尽可能反映生产环境，但你也需要保持它们的物理隔离，否则就得承担后果。
>
> 为了将生产环境和测试环境隔离，你应该考虑为测试环境使用不同的颜色标示。你不应该把生产环境连接到测试环境中，反之亦然。此外，很关键的是要标记测试环境，这样就没有人会在生产环境中使用它们。

9.2.4 使用虚拟机

大部分公司没有足够的硬件来为应用程序用户所依赖的每种操作系统和浏览器配置单独运行一台机器。解决方法就是使用虚拟机，这样你就能通过配置一台计算机来运行多个虚拟

机。每个虚拟机会代表用于测试的一种用户配置。

使用虚拟机还有另外一个理由。当虚拟系统被你制造的攻击击垮时，你可以简单地停掉它，删除相关文件，并从保存在硬盘中的基线配置中创造出新的副本。你可以在几分钟内就搭建出新的测试系统，而不需要花上几小时。

虚拟机可以解决搭建系统中的许多问题，但你也需要考虑高端硬件的需求以使其可正常工作。动力不足的系统无法产生可用于评估安全问题的结果。你能在一台物理机器上创建的虚拟系统的数量受限于内存大小、处理周期以及在这一过程中系统必须使用的其他资源。

虚拟机操作软件能够提供你想要测试的各种平台，这也很关键。VMWare（http://partnerweb.vmware.com/GOSIG/home.html）等产品可以支持大部分主流操作系统。当然，这种支持会有额外的开销。

有些公司会采用仍然可以使用的旧系统来做测试，这是虚拟化的一种替代方案，但这些系统并没有它们曾经在生产环境中时那样有用。由于 Web 应用程序工作方式的特点，这些旧系统通常可以提供所需的各种计算能力并让公司继续从旧设备中受益。当然，维护这些旧系统也会有一笔开销，所以你需要权衡使用虚拟机及使用旧系统的成本差异（或者可以将两者相结合）。

9.2.5 获取工具

除非你想要写自己的安全工具软件（一个毫无疑问的坏想法），否则就需要从别处获取它们。幸运的是，McAfee（http://www.mcafee.com/us/downloads/free-tools/index.aspx）等网站会提供各种免费工具，你可以用它们来执行以下任务。

- 检测主机系统里的恶意软件
- 评估系统是否易于遭受攻击
- 在遭受一次攻击后执行诊断分析
- 使用 Foundstone 的软件应用安全服务（Software Application Security Services，SASS）工具让应用程序更安全
- 确定入侵发生的时间
- 扫描系统的各种弱点
- 对系统进行压力测试

9.2.6 配置系统

从一个干净的配置开始搭建系统对于遭受攻击后的系统进行诊断测试是很重要的。在你安装任何软件或执行任何配置操作之前，请确保你用零做了擦除（在各个地方写上零）。许多软件产品会让你执行这一任务。将磁盘写满零可以确保你看到的任何数据都是你测试的结果，而不是前一次安装遗留下来的信息。

从一开始就为你想要支持的每个平台创建一个干净的配置是很重要的。请确保副本是你从配置镜像中弄出来的，这样你就能在以后恢复它。配置应该包含操作系统、浏览器，以及任何支持测试环境所需要的测试软件。你也可能需要包含用户通常会安装在系统中的软件

（如果 Web 应用程序会以任何方式与其交互）。

> Norton Ghost（http://www.symantec.com/page.jsp?id=ghost）等产品可以非常容易地创建每个配置的镜像。在你对自己创建的干净配置做任何事之前，请确保脑中有一个镜像创建策略。随后当你通过攻击破坏了配置之后，将需要干净的镜像来恢复配置。

除了标准的软件，你可能需要安装远程访问软件，这样你可以在物理安全的测试区域之外远程访问系统。外部访问必须使用物理隔绝的网络来确保没有污染生产环境的风险。远程访问软件的使用可以让一名以上的测试者从他们日常的办公场所根据需要访问系统，而不必从物理上安全的区域进行访问。

> 有些公司用 KVM（键盘、视频和鼠标）交换机提供可访问测试系统的工作站。采用一个 KVM 配置和一个传输器就可以从远程轻松切换多个测试系统。但是，使用远程访问软件可能会更灵活和快捷。

9.2.7　恢复系统

用不了多久你的测试系统就会需要恢复。你用来测试的病毒、广告软件和木马只是其中的一个问题来源。执行入侵并确定破坏应用程序的难度终究会损坏操作系统、测试应用、测试数据以及用于保护系统的对策。总之，你需要一种方法来快速将系统恢复到某个已知状态。

恢复还包括重新配置系统以模拟另一个环境。使用镜像来快速创建各种测试环境是值得的。如前面提到的，使用虚拟机会节省大量时间，因为你不必每一次都从头开始重建硬盘驱动。但是请确保你在其他的硬盘、DVD 或其他存储介质中也有每个操作系统的镜像，因为硬盘终究会遭到损坏。

9.3　定义最常见的漏洞源

每天都有新的安全漏洞出现。任何人都不可能跟进所有的漏洞。但是，某些安全漏洞需要特别关注，而其他漏洞可能未来才会碰到。当然，最好是有一套有条理的方法来定位安全漏洞，而 OWASP 提供的清单（https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet）无疑是一个正确的开始。

一旦你了解了应用程序可能存在的攻击途径，就可以对它们进行打分，这样就能知道哪些漏洞需要优先修复。一种能让他人访问公司所有野餐照片的攻击，其修复优先级要远远低于让别人访问客户信用卡信息的攻击。用来对潜在的攻击途径进行打分的最好系统之一是通用安全漏洞评分系统（Common Vulnerability Scoring System，CVSS，http://www.first.org/cvss/v2/faq）。使用这个系统可帮你创建一份有条理的问题清单，你可以根据它来进行处理，从而使你的应用程序更加安全。

当然，它还会帮你弄清楚要测试什么。为此，接下来会描述本书撰写时最常见的安全漏洞

源。（这以你从第 1 章获取的关于基本攻击的信息为基础。）当本书出版时你可能会发现更多的安全漏洞，因为如果在破坏软件方面没有创造力，黑客就不是黑客了。

> 你无法对黑客会使用的每一种可能的漏洞源进行分类。事实上，黑客常常利用误导（非常像魔术师）来阻止你意识到正在发生什么。一篇最近的关于波兰航空公司（LOT Polish airlines）的新闻（http://www.computerworld.com/article/2938486/security/cyberattack-grounds-planes-in-poland.html）可用来说明误导的后果。在这个例子里，当局花了大量时间和资源来确保飞行系统不会受到潜在攻击的影响。但是，他们并没有在地面系统上花太多力气。黑客设法入侵了地面系统并使其无法为出国乘客创建飞行计划。这个攻击是通过影响飞行系统还是通过影响地面控制来让飞机停飞已经不重要了，重要的是飞机无法起飞。黑客通过误导达到了目的。这个例子告诉你需要查看每一处地方，而不仅仅是根据最近的统计得出的黑客会攻击的地方。

9.3.1 避免SQL注入攻击

SQL 注入攻击有多种形式。例如，你会处理应用程序中的表单数据来确定该表单是否能够发送命令到后端服务器。但是，最好是首先有一个检查潜在 SQL 注入攻击的通用方法。

假设你有一个像 http://www.mysite.com/index.php?itemid=10 这样的 URL。你会在很多网站上看到类似结构的 URL。检查漏洞的一个方法是简单地在 URL 后面加上一个单引号，使其成为 http://www.mysite.com/index.php?itemid=10'。当你按下回车，网站会发送回一段 SQL 错误消息。这个消息会根据系统有所不同，关键是后端服务器会接收你的 URL 作为 SQL 请求，它会被拼成类似这样的格式：`SELECT * WHERE itemid='10''`。多出来的单引号会使查询不合法，从而产生错误。

> 当你真的在自己的应用程序中发现一个别人可以利用的问题时，很重要的是要让整个开发团队认识到它。把问题展示出来，让每个人都能看到它是如何工作的以及需要做什么来修复它，这对于任何公司来说都是开发人员训练中很重要的一部分。浏览器攻击框架（Browser Exploitation Framework，BeEF，网址为 http://beefproject.com/）等产品可帮你找出攻击途径，然后展示给公司的其他人。

你现在可以开始操作 URL 来看看会发生什么。例如，你想看看查询会产生多少列数据以及这些列都是什么。可以使用 `ORDER BY` 这样的 SQL 语句来执行这一任务。改变一下 URL 使其包含 `ORDER BY` 语句，就像这样：http://www.mysite.com/index.php?itemid=10' ORDER BY 1。这与输入 `SELECT * WHERE itemid='10' ORDER BY 1` 这样的命令是一样的效果。通过在每次请求中给 `ORDER BY` 的值加 1，你最终会看到一个错误。假如当你尝试到 `ORDER BY 10` 时看到了错误，那么可以确定在这个例子中请求会产生 9 列数据。

黑客会继续在 URL 中加入 SQL 命令来了解更多数据库的查询结果。例如，使用 `SELECT` 语句可以帮你确定哪些数据会出现在屏幕上。你也可以请求特殊的信息作为 `SELECT` 语句的一

部分，比如使用 @@version 来获得 SQL 服务器的版本号（会给你一些关于该 SQL 服务器可能存在缺陷的信息）。这里的关键是原始的 URL 可以直接访问 SQL 服务器，使得别人完全不需要做太多工作就可以控制 SQL 服务器。

你可以在网页 http://www.w3schools.com/sql/sql_injection.asp 上看到另一种 SQL 注入攻击。这些攻击根本的原因是开发人员直接使用来自页面或请求中的数据，而没有首先检查其是否有问题并将错误信息移除。

9.3.2 理解跨站脚本攻击

XSS 在攻击方式上与 SQL 注入有很多相似之处，但实际上这两种技术是不同的。XSS 有两种类型：非持久的（攻击依赖用户访问某个专门制作的链接）和持久的（攻击代码保存在间接的存储媒介中，比如数据库）。这两种攻击都依赖 JavaScript 代码被放到你不希望它们出现的地方。

作为一个非持久 XSS 攻击的例子，可以看看链接 http://www.mysite.com/index.php?name=guest。它看起来像是一个完全无害的链接，带有一个名 / 值对。但是，如果你将一段脚本加入到其中，比如 http://www.mysite.com/index.php?name=guest<script>alert('XSS')</script>，用户会看到一个对话框弹出来，带有信息 XSS。这个例子不会造成任何破坏，但这个脚本可以轻易地做任何你在脚本中可做的事情。例如，你可以定制脚本，让它把用户重定向到另一个网站，那个页面会下载一个病毒或其他恶意软件。

> 这个例子用直接的文本展示了 <script> 标签。实际的攻击会对 <script> 标签编码，这样用户就不能轻易地识别出它。用户看到的会是一个很长的 URL，带有一些很复杂的 % 值，就像平时在合法的 URL 中看到的那样。

一个持久的 XSS 攻击会比较难实现，但会造成更大的破坏。例如，想象用户登录应用程序之后会发生什么。服务器会将一个会话 ID 作为 cookie 发送回用户的机器。之后的每次请求都会使用这个会话 ID，这样应用程序就能维持这个用户的状态信息。假如攻击者将经过特别定制的脚本发送给服务器作为登录过程的一部分，那么它会被保存到数据库中，用的名称是管理员几乎肯定会去查看的名称。当管理员点击用户名时，这个脚本会执行并会将管理员的会话 ID 发送给攻击者。攻击者现在拥有了能操作其他会话的管理员权限，他可以执行任何管理员级别的操作。你可以在网页 https://www.acunetix.com/blog/articles/persistent-cross-site-scripting/ 上获得更多关于持久 XSS 的详细信息。

在这两种情况下，防御 XSS 的最佳方法就是对任何你接收到的输入进行净化。净化输入的过程会移除任何脚本、怪异的字符或其他不期望出现在响应中的信息。例如，当你期望一个数字型的输入时，响应不能是包含有字母的字符。

> 输出经过编码的用户数据有助于减少潜在的 XSS 风险。黑客仍然可以绕过编码，但需要额外的时间和精力来这样做。你所做的任何会提高攻击难度的事情都会增加黑客转移目标的愿望，但是你还要记住，一名坚定的黑客总是能破坏你的安全。

9.3.3 解决拒绝服务攻击问题

DOS 攻击的思想是相对直接的。你找到服务器上的一个开放端口并持续发送没有意义的包给它，从而压垮相关的服务。大部分服务器都有提供开放端口的服务，比如以下这些服务器。

- DNS 服务器
- Email 服务器
- FTP 服务器
- Telnet 服务器
- Web 服务器

当然，你开放越多的端口，服务器被压垮的机会就越大。防御的第一条是通过下线不需要的服务来关闭不需要的端口。一个应用服务器可能只需要作为 Web 服务器，那么这就是你唯一应该安装的服务。作为一名应用开发人员，你要建议下线其他服务（至少不要启用它们）。当创建某个私有应用程序时，采用非标准的端口也是有帮助的，比如请求认证的时候。

黑客通常会寻找没有最大连接数的服务，所以要确保最大连接数是服务器能处理的值，这是另一个正确的步骤。DOS 攻击中最重要的是让服务器一直运作，比如让 Web 服务器执行一次复杂的搜索。认证机制有助于防止黑客在没有适当认证时发起请求。

还有，如果有足够多的系统在发送无尽的没有价值的请求，也可能把服务器拖垮。一些抵挡 DOS 攻击的措施包括寻找该请求可匹配的模式然后简单地拒绝它们，而不是花费系统资源去处理这些请求。你可以在网页 http://resources.infosecinstitute.com/dosattacks-free-dos-attacking-tools/ 上找到大量 DOS 攻击工具来测试你的系统。此外，除了自己去查找攻击的匹配模式并解决它们，你还可以尝试采取以下措施。

- 购买专门设计用来对抗 DOS 攻击的设备
- 依靠网络服务提供商（ISP）来检测并消除 DOS 攻击
- 获取云缓存提供商的服务

9.3.4 去除可预测的资源定位

通过了解特定资源的位置，然后使用这些资源来获取访问系统的足够信息，是有可能对系统发起攻击的。有些网站还将这种攻击称为强制浏览。当然，防止这种攻击的最佳方式是将资源放在不可预测的位置。此外，你要确保认证体系是有效的并且资源是安全的。无论如何，让我们看看这种攻击是如何发生的。

这种攻击的一个例子是，识别一个指向合法资源的 URL 并使用这个 URL 去访问属于另一个人的资源。假如你在网页 http://www.mysite.com/Sam/10/01/2015 上保存了你某一天的日程安排，而你想要看看 Amy 在同一天的日程安排。如果管理员没有正确配置服务器的认证，将 URL 改为 http://www.mysite.com/Amy/10/01/2015 就有可能可以正常访问到该信息。

另一个例子是，某些服务器会把信息放在特定的目录下。例如，你可能认证了对 http://www.mysite.com/myapp/app.html 的访问，但是你可以将 URL 改为 http://www.mysite.com/system/ 来看看其是否存在。如果你从服务器得到一个 200 的响应，那么这个目录就存在，

你就可以开始请求它来获得有用的信息。当然，这里假设管理员没有给系统目录加上适当的安全措施，并且 system 是一个标准的目录位置。管理员总是可以更改系统目录的名称，还可以确保只有拥有适当凭据的人员才可以访问它。

9.3.5 克服无意的信息泄露

无意的信息泄露会以各种方式发生。但是，这类攻击通常涉及黑客未经授权地访问某个集中式数据库。以前，信息源是网络信息系统（Network Information System，NIS）这样的东西。但是，今天的信息可能有任何来源，这些来源不够安全或有黑客可利用的缺陷。这样的源头有很多，实在不可能用一个简单例子将它们都展示出来。你可以通过以下措施来克服这类问题。

- 给操作系统、服务、应用环境、库、API 和微服务打上必需的补丁
- 配置边界路由器和防火墙来阻止从敏感来源发起的信息请求
- 限制对所有敏感信息源的访问
- 永远不要硬编码密码并把它们放在别人能轻易找到的地方
- 对任何敏感信息源使用双因素认证
- 执行审核来查找潜在的漏洞（即使你感到系统完全安全）
- 使用评估工具来确定是否存在从指定位置外的任何位置访问敏感信息源的可能

9.4 在BYOD环境中进行测试

BYOD 现象在持续增强。重要的是要认识到 BYOD 不会消失。事实上，你可能已经认识到 BYOD 在某些方面正在变成给用户提供设备的最佳方法。公司最终会告诉用户携带任何完成工作所需的设备，并且把所有东西都交给用户。这听起来像是一场灾难，但这是用户现在正在做的事情。

根据 Gartner 公司的报告，到 2017 年，一半的公司将不再提供任何设备给用户（请参阅 http://www.gartner.com/newsroom/id/2466615）。此外，到 2020 年，75% 的用户会用不到 100 美元来购买一部智能手机（请参阅 http://www.gartner.com/newsroom/id/2939217），所以获取一部足够智能的、能完成大部分任务的设备不会很昂贵。创建并管理应用程序会更加困难，因为你必须确保应用程序确实能在任何地点和任何设备上运行。当然，它还不能破坏公司的安全。接下来会帮你为 BYOD 环境提供某些级别的测试和隔离。

> 还有一点很重要，要认识到用户现在越来越倾向于使用自带应用程序（bring your own application，BYOA）。一个非常重要的原因是，它给应用环境引入了另一种级别的不可预测性。你永远无法知道另一个应用程序何时会使你的应用程序出现问题。事实上，第三方应用程序会为你的公司遭遇的下一个主要漏洞提供渠道。用户肯定会继续使用 Dropbox、Google Docs 和 CloudOn 等应用程序，因为它们很方便且能在任何地方运行。为了确保应用环境的完整性，你需要继续将这些应用程序看作正在等着摧毁你的污染物。

> BYOA 是一个问题，其中一个原因是公司会失去对数据的控制。如果用户将公司数据保存在 Dropbox 的个人账户里，公司不能轻易地在发生紧急情况时找回这些数据。总之，BYOA 打开了严重的安全漏洞，这可能对你想要保护的任何应用数据都是一个问题。

9.4.1 配置远程访问区域

当在 BYOD 环境中工作时，最好假设设备环境是不安全的且无法简单地使其变得安全。为此，BYOD 会话通常有以下 4 个阶段。

(1) 客户端和服务器端创建一个安全的管道以使得外部难以窃听。其目的是防止中间人攻击。
(2) 用户提供两种形式的认证。一个双重认证过程会使别人不太可能成功地模拟该用户。
(3) 客户端发起一个或多个请求，而服务器端使用一个服务中转模块来进行中转。服务中转模块只处理对合法服务的请求。服务中转模块会自动记录每次请求是成功还是失败。
(4) 服务隔离模块只提供对公共数据的访问。它不允许对敏感数据进行访问。客户端只能看到公司认为在 BYOD 环境下可接受的数据。

远程访问区域属于阶段 1 和阶段 2 的部分。它包括一个外部防火墙和一个 VPN 以及认证网关。远程访问区域提供了对抗入侵的第一级别的防卫。

从用户那里收集的信息必须体现为你的应用程序策略的一部分。一次 BYOD 访问不同于公司桌面系统发起的本地访问，你无法知道访问发生在何处或设备本身是否安全。当用户以这种方式访问应用程序时，你需要提供一种匹配所使用设备的角色。这意味着你不能让任何敏感数据出现在这个设备上且可能限制对其他数据的访问。例如，你的公司可能认为让某个客户端获得销售列表是可接受的，但这个列表是只读的，这意味着用户不能对销售列表做任何改动。为了做出更改，用户需要从本地系统登录账户。

使用远程访问区域也意味着公司要配置移动设备管理（mobile device management，MDM）。这是有助于确保移动设备尽可能保持安全的一些产品和服务。例如，MDM 会检查移动设备的补丁要求，并确保在访问应用程序之前给设备打上补丁（你可以自动为设备打上补丁）。就算有 MDM，当出现以下情况时，你也不能假设设备是安全的。

- 设备报告了一些与安全配置有关的值，但你的公司实际上并没有实现这些特定的值。一个恶意软件的开发人员不会知道哪些值需要上报，所以会简单地将它们全都上报。
- 其他机构会覆盖设备的设置。例如，一个智能手机供应商会在你的控制之外自动给设备打补丁，你必须假设其中一些补丁会包含病毒或其他恶意软件。
- 不可能给移动设备强加设置。设备可能不提供所需的支持，可能没有将应用程序配置为支持该设备，或者有恶意软件干扰其更新。

9.4.2 检查跨应用程序的攻击

跨应用程序攻击是指一个应用程序获取另一个应用程序使用的数据或资源。在许多情况下，攻击发生于两个应用程序访问相同的数据源时（跨应用的资源访问或 XARA）。最近的一次这种攻击是针对 OS X 和 iOS 操作系统的（请参阅网页 http://oversitesentry.com/xara-

an-old-way-to-hack-cross-application-resourceaccess/，获取相关资料）。

另一种类型的跨应用程序问题是跨应用脚本（cross-application scripting，CAS）。在这种情况下，JavaScript 代码在某些类型的应用程序中执行时，bug 会导致问题。你可以在网页 http://news.softpedia.com/news/Apache-Cordova-for-Android-Patched-Against-Cross-Application-Scripting-453942.shtml 上查看针对 Android 的这种攻击。

验证你的应用程序不会在这种攻击中泄露数据的最佳方式是持续关注你要支持的平台的新闻。你需要一名安全专家来找出潜在的这类问题。遗憾的是，如果禁止从操作系统或浏览器供应商中获取补丁，你就不能真正对这种攻击做太多事情，只能保持警惕，检查数据存储可能遭到的破坏。当然，这种问题只会给创建远程访问区域提供更多的凭据（见本章前面的部分）。

9.4.3 处理真正古老的设备和软件

研究机构喜欢描述美丽的蓝图，讲述未来看起来会是什么样子。当然，其中一些梦想真的会实现，但他们不会考虑当前在用的用户设备的规模的现实。例如，在发现（在本书编写时）95% 的 ATM 机还在使用 Windows XP 操作系统时（http://info.rippleshot.com/blog/windows-xp-stillrunning-95-percent-atms-world），某些人可能会感到麻烦。美国海军仍然在 100 000 台桌面电脑上使用 Windows XP（http://arstechnica.com/information-technology/2015/06/navy-re-upswith-microsoft-for-more-windows-xp-support/）。是的，旧的、古老的、破旧的软件仍然存在，并且你可能发现它们还被用来运行你的应用程序。

根据 NetMarketShare（http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcu- stomd=0）的数据，Windows XP 仍然有高达 14.6% 的系统占有量。重要的是要认识到当你创建一个应用程序时，应该主要关注的可能不是得到更新的闪亮的新智能手机、完全打了补丁的操作系统以及最新的浏览器技术。真正需要关注的重点可能是老旧的 Windows XP 系统和用户坚持使用的 IE8 浏览器。很有趣的是，IE8 仍然有大约 25% 的桌面市场占有量（https://www.netmarketshare.com/browsermarket-share.aspx?qprid=2&qpcustomd=0）。

当然，你总是可以尝试强迫用户升级。但是，如果你已经处理 BYOD 现象足够长的时间，你会知道用户只会忽略你。是的，他们可能会展示给你一个闪亮的新系统，但他们会继续使用他们喜欢的系统，而那些老旧的东西会导致应用程序遭受许多灾难。

可能解决过时设备问题的唯一方法是在请求的过程中检查浏览器数据。这样做可以让你选择是否让请求成功发送。当浏览器非常旧时，你可以显示一条消息，通知用户去升级他们的设备。网页 http://sixrevisions.com/javascript/browser-detection-javascript/ 上的一篇文章描述了如何执行浏览器版本检查。你可以在网页 https://www.cyscape.com/showbrow.asp 上看到自己的浏览器信息。这个方法的唯一问题在于有些时候检查可能失败，并且有些浏览器还会错报版本信息。

9.5 依靠用户测试

没有谁能用比用户更暴力和更不可预测的方式来测试软件。大部分开发人员已经发现用户

会尝试让软件做一些它们明显不能做的事情，因为用户不知道软件不能这样做。软件明显能做的事与用户让它做的事无关，这是只有用户才能够做的严格测试。

这个过程的重要之处在于它会给开发人员带来最深的痛苦。任何尝试使用用户测试变得有条理的做法都会让测试失败。用户需要自由地操作软件。事实上，游戏时间对于每个人都是好的，但它是用户测试的一个关键元素。接下来会描述你应该如何让用户操作你的软件，并发现似乎只有用户（显然还有黑客）才能发现的严重安全漏洞。

> 有趣的是，你再也不必执行自己的用户测试。如果你真的想要一个广泛的测试用户基础，但又没有资源这样做，那么通常可以使用 User Testing（http://www.usertesting.com/）这样的网站。理论上，这个网站可以在短短 1 小时内（虽然其主页上说可以在 20 分钟内交付结果）提供关于你的应用程序的客观评价。
>
> 一些第三方测试商，比如 Applause（http://www.applause.com/web-app-testing），专门提供“广泛的”测试，你的应用程序真的会在互联网上被未知的用户使用，就像其在生产环境中一样。当然，不同点在于进行的测试不会对你的设备产生任何风险，而 Applause 会提供所需的工具来获取可测量的结果。一般说来，你要等到应用程序准备发布且你想要执行一次完整性检查时，再进行这个级别的测试。

9.5.1　让用户横冲直撞

有时候，测试应用程序的最佳做法是给用户一些打印的步骤和一个目标。看着用户如何与应用程序交互，你会获得很多关于用户如何看待你的应用程序以及你需要在哪里做出改变的信息。事实上，用视频录制用户的操作会有所帮助，但你需要认识到录像行为会改变用户的行为。键盘记录以及其他的技术也有助于记录用户操作应用程序的踪迹，而不需要站在用户的上方观察他们（这势必会改变用户的行为且你的测试会失败）。

幸运的是，你不必仅仅依靠自己手头上的资源。Mashable（http://mashable.com/2011/09/30/website-usability-tools/）等网站会提供很多可用来检测 Web 应用程序问题的攻击方法。这个网站有很好的有关工具的文档，并会告诉你为什么每个工具都很重要。最重要的是，这个网站会帮你理解某些你可能从来没有考虑过的测试的重要性。例如，Check My Colours（http://www.checkmycolours.com/）会验证有不同视觉需求的人是否能真的看清你的网站。是的，使用错误的颜色真的会是一个问题，而测试这个问题会帮你避免潜在的、由用户的错误导致的安全问题。

另一个可找到测试工具的好地方是 The Daily Egg（http://blog.crazyegg.com/2013/08/08/web-usability-tools/）。这个网站上的某些工具与 Mashable 网站上的是一样的，但你可以获得更深入的理解。有一些工具是独家的。例如，在 The Daily Egg 上的页面速度工具清单就比较好，这份清单包含有 GTMetrix（https://gtmetrix.com/），它可帮你定位页面中真正慢的那部分代码。

9.5.2 开发可重现的步骤

用户测试的一部分是要获取一些可重现的步骤。除非你获取了这类信息，否则就不能真正解决导致问题的应用缺陷。这就是你需要特定的测试工具来记录用户详细操作过程的原因。让用户重现产生错误的步骤永远都不会奏效。在许多情况下，用户并不知道错误是如何产生的，并且只会感到计算机不喜欢他们。尝试让用户重现步骤很可能会让他们生气，并使他们不喜欢你的应用程序（这通常会导致更多错误和更多安全问题）。因此，你需要在首次尝试时就获取重现错误所需的步骤。

为了创建一个可获得最佳测试水平并确保用户有很大机会发现潜在错误的环境，你需要执行特定类型的测试。让用户横冲直撞来发现在没有任何有用的输入时他们会做什么，但混乱很难产生可预测的结果。当进行用户测试时，你需要考虑以下这些类型的测试。

- 功能性
  测试所有的应用程序功能是很重要的。这意味着要让用户尝试表单、执行文件操作和计算任务，使用应用程序功能来搜索信息，并且尝试应用程序特性提供的媒体功能。当用户测试这些功能时，请确保测试也检查应用程序用来执行大多数任务的库、API 和微服务。

- UI 和可用性
  UI 必须保证让用户感兴趣并为任何有特殊需要的人提供支持。作为这一级别的测试的一部分，你需要检查导航、可访问性、多个浏览器下的可访问性、错误消息和警告、你提供的帮助和任何其他文档，以及布局。

- 安全性
  虽然你已经测试过应用程序以确定其是否有 9.3 节列出来的常见安全漏洞，你还是需要让用户来测试它们。看看用户是否会用与你相同的方式使应用程序受到破坏。找到可产生新的漏洞条件的用户操作方式。

- 负载与可伸缩性
  你不可能完全测试一个应用程序在高负载下的表现。应用程序的性能会随着负载的增加而降低，你需要确保其能够很好地伸缩。但是，最重要的是，你需要验证负载不会导致安全事故发生。无论负载有多少，应用程序都能继续工作，知道这一点很重要。当然，当负载超过预期时，应用程序会运行得慢得多，但这与真正的破坏是不一样的（例如，放某人进来的事故）。

9.5.3 让用户发声

在用户使用应用程序之后对其进行采访，或允许用户对应用程序进行抱怨，这是发现更多潜在问题的好途径。在许多情况下，由于预期应用程序会被中断，用户会害怕执行某些步骤。这个问题甚至在测试环境中也会发生，用户在测试环境中就应该去破坏应用程序。你可能不想听到用户说什么，但在开发阶段听到用户的反馈比把应用程序放到生产环境后再发现严重问题要好得多。

收集用户反馈的另一种方法是使用问卷调查。一次匿名的问卷调查会帮你获取用户可能羞于提供的信息。重要的是要考虑压力和预知风险对你接收到的用户反馈的影响。

9.6　使用外部的安全测试人员

渗透测试要依靠第三方的服务来确定一个应用程序、网站或甚至是整个公司是否易于遭受各种入侵。攻击者会采用与黑客相同的技术来探测防御措施。由于攻击者是安全方面的专家，测试水平可能会优于公司自身提供的测试。大部分公司会因为以下这些理由而使用外部安全测试。

- 定位内部审核时遗漏的缺陷
- 给客户和第三方提供一份对应用程序安全性的独立评审
- 在公司没有任何安全专家的时候要确保安全测试是完整的
- 验证事件管理程序的有效性
- 培训事故处理团队
- 降低整个公司的安全成本



在没有保密协议（nondisclosure agreement，NDA）等法律文书的情况下允许别人对你的应用程序、网站或公司进行渗透测试是一个不好的做法。不要假设你可以相信任何人，特别是那些测试安全性的人。请确保所有东西都有文字记录，这样从一开始就没有误解的机会。

9.6.1　考虑渗透测试公司

当然，与其他事情一样，渗透测试会带来一些风险。例如，你可能期望第三方测试人员可以完全诚实地向你报告在应用程序中发现的缺陷，但通常并不是这样。在某些情况下，第三方并不会提供完整的缺陷文档，但在其他情况下，渗透测试人员实际上可能会估量你的公司以确定发起一次真的入侵是否是一个好主意。重要的是要确保你用的是一家声誉良好的安全测试商并验证其展示的服务类型。如果你有以下这些经验，应该花更多的时间检查你的安全服务。

- 在渗透测试阶段发生的敏感数据泄露、滥用或丢失
- 遗漏的缺陷或弱点
- 目标应用程序或网站的可用性受到的影响
- 测试没有及时进行
- 报告过于技术性并难以理解
- 项目管理看起来杂乱无章、匆忙，或考虑不周

当谈到渗透测试时，你付出多少就会得到多少。在聘请第三方进行渗透测试时，公司必须权衡成本、时效性和测试质量。你付出的越少，等待得就越久，获得的也会越少，测试后发生负面情况的可能性也会越大。

在某些情况下，减少测试工作量要比其他取舍的效果更好。只测试一个应用程序而不是整个网站，开销就会小，而你可以聘请更高质量的安全机构来执行这一任务。当然，如果你只测试一个应用程序，就不能检查目标与其他系统之间的信任关系等问题。

9.6.2 管理项目

在允许任何人执行渗透测试之前，你需要一份提案来划出范围、目标和测试方法。任何渗透测试都必须包含社会工程学攻击，因为大部分黑客都会利用它们。事实上，公司人员是安全性中最薄弱的环节。一个应用程序可以提供近乎完美的安全性，但由犯错的人员导致的一个泄露会破坏你在维持安全环境方面的所有努力。

请确保测试方法有很好的文档记录并遵循业界的最佳实践。例如，许多安全机构遵循开源安全测试方法（Open Source Security Testing Methodology，OSSTMM）；可以到安全与开放方法研究所（Institute for Security and Open Methodologies，ISECOM）网站（http://www.isecom.org/）查看更多细节。如果测试人员使用了其他方法，请确保你准确理解了这种方法以及它会提供什么好处。

该提案应该陈述你要获得什么类型的反馈作为测试的一部分。例如，很重要的是一点是，要决定渗透测试需要精确说明系统是如何被渗透的，还是只需要报告某个区域易于受到攻击。

定义攻击时间也很重要。你可能不想在一天中最忙碌的时间段进行测试，以避免对销售造成风险。但另一方面，如果你真的想要看看负载对系统的影响，可能就需要在不太适合的时间段进行测试。

9.6.3 覆盖要点

你感兴趣的任何渗透测试公司都应该有一个单一联系点。你能够在任何时候联系上他们是很重要的。如果有需要，开发团队与渗透团队的负责人应该能够在任何时候联系到对方以停止测试。这种联系还应该包含完整的细节，描述测试如何进行，以及在当前的测试阶段可能发生的任何潜在问题的细节。

你还应该知道测试者是否有覆盖测试阶段发生损坏的责任保险。责任保险应该包含对修复数据花费的时间、系统停机时间以及公司可能遭受的任何潜在损失的赔偿。

测试团队还必须证明其能胜任工作。你不希望随便什么人都去渗透测试你的系统。出现在提案中的团队也应该是实际进行测试的团队。检查他们是否有以下这些证书。

- GIAC 事故处理认证（GIAC Certified Incident Handler，GCIH）
- 道德黑客资格认证（Certified Ethical Hacker，CEH）
- OSSTMM 专业安全测试人员（OSSTMM Professional Security Tester，OPST）

9.6.4 获取报告

任何渗透测试的结果都要体现为一份报告。为了确保你获得一份真正可用的报告，你一定要在测试开始前要求一份样例报告。该报告应该包含测试概要、任何漏洞的细节、风险评估以及渗透测试所涉及的所有行动的细节。报告必须包含足够的技术信息，使你确实可以修复在测试中发现的问题，但仍然要能被管理人员看懂，这样你才能获取所需的时间和资源来执行修复。

除了报告，你还应该接收在测试阶段执行的每个动作的日志文件。日志文件应该展示在测试过程中每个包的发送和接收，这样你可以在后面一步一步地重现测试过程。

你雇用的安全公司也应该归档测试过程。你需要知道他们计划如何保存这份档案。检查归档过程的一个原因就是你要确保你的私密数据是安全的。离线存储而不是在供应商的网络硬盘中进行归档是很重要的。

第10章

创建API安全区域

你在应用程序中创建或使用的任何 API 都有可能带来大量问题。但是，与库不同，你确实可以让 API 的使用变得更加安全，因为 API 在自己的地址空间和自己的进程内执行。将 API 放在沙盒或虚拟环境（特别是受保护的环境）中可以实现以下控制。

- 精确地控制 API 执行的动作、访问的资源和它与应用程序交互的方式。当然，你也可能令 API 得不到所需资源，或者使沙盒或虚拟环境兼容过多东西而令 API 不能完成任务。你必须在风险（安全）和有效工作之间维持平衡。
- 控制应用程序如何与 API 交互。例如，你可以减小错误或恶意的输入导致严重灾难的可能性。严格控制应用程序的输入，非法输入造成的影响就会减少，甚至完全没有影响。当然，这种保护也让使用 API 或执行某些测试变得困难。

本章会帮你理解 API 沙盒 / 虚拟环境的概念，并明确你在下一个编程项目中该如何使用它们来保证安全。作为使用 API 沙盒的一部分，本章还会探讨一些沙盒和虚拟环境的解决方案。不是每种解决方案在所有条件下都能起效，所以多知道一些潜在的解决方案是很好的。

> 与任何其他安全解决方案一样，使用 API 沙盒不会是一个捷径。它不会让每个黑客都无计可施，并且也不能完全阻止 API 对应用程序或相关数据的破坏。使用 API 沙盒会降低风险。你需要确定 API 沙盒是否能降低足够多的风险，或者你是否需要额外的方法来保证应用程序安全。安全不只与一门技术或一个方法有关。一个安全系统要依赖许多安全层来完成工作。

本章的最后一节会讨论沙盒的一个替代方案——虚拟化。尽管虚拟环境看起来与沙盒是本质上相同的东西，因为它们达到的目标是大致相同的，但它们确实是不一样的技术，因此最后一节会帮你详细了解二者的不同点。虚拟环境让 API 在一个完全开放的环境中执行。但是，虚拟环境与其他任何东西是完全隔离的，并且 API 会有严格的访问策略。

本章的要点是帮你创建一个环境，在这个环境中 API 可以在不与应用程序的其他部分相关联的情况下执行。这个环境是受到严格控制的。本章展示了达到这个目标的两种方法，并帮你了解在什么时候应选择哪一种方法。如果你想要从应用测试以及最终的使用中获得有用的结果，选择正确的安全区域是很关键的。

> 本书使用术语安全区域来指代沙盒和虚拟环境所要达到的目标。关键是要记住这两个技术的目标是基本一致的，但这两个技术达成目标的方式是非常不同的。因为这些不同点，你可能会发现选择其中一个会给你的某些应用开发需求带来好处。

10.1　理解API安全区域的概念

API 安全区域会提供一个安全且灵活的方法来测试或使用你创建的 API。在测试阶段使用一个 API 安全区域以确保可以快速地从错误中恢复，通常是一个好的做法。因为其所提供的安全特性，你可能会在把 API 放到生产环境中之后决定继续使用 API 安全区域。

使用沙盒确实能提供额外的安全性。但是，许多公司在实现沙盒环境时并不关心安全性。除了安全性，沙盒还可以提供以下好处。

- 通过控制资源使用而降低成本
- 受控制的第三方 API 访问
- 用更好的控制来使测试和开发环境获得提升
- 缩短推向市场的时间
- 为测试模拟错误场景
- 受监控的 API 性能

公司也会使用虚拟环境来达到其他目的，而不只是使 API 变安全。与沙盒一样，一个虚拟环境会通过控制资源的使用来降低成本，控制对第三方 API 的访问，以及改善测试和开发的环境。从公司的角度来说，虚拟环境有以下优点。

- 缩短 API 崩溃之后的恢复时间
- 增强速度和网络带宽控制
- 改进能源的使用
- 减少硬件的占用空间
- 更快的物资供应
- 减少供应商锁定的情况
- 增加正常运行时间（即使在安全性和可靠性事件很少时）
- 延长应用程序的生命

如果使用沙盒和使用虚拟环境时 API 都能很好地工作，那么究竟是选择使用沙盒还是使用虚拟环境，最终可能会归结为它们所能提供的额外功能。但是，如果你想维持如今应用程序要求的安全性，很重要的是要首先考虑这两者在安全方面的表现。

10.2 定义API安全区域的需求

安全就是风险管理。请求任何 API 都会带来风险，因为你不知道这个 API 有没有被损坏，而是首先假设这个 API 是没有潜在问题的。当你在网上读到关于 API 带来的伤害的信息时，你对所使用的 API 进行管控的需求就会变得明显。你使用的任何 API 都可能发送经过伪装的数据，它们实际上可能是病毒、脚本或其他恶意软件。由于 API 的使用方式，你只能在每个人都开始抱怨应用程序无法工作或你在不希望的地方发现机密数据（假设其他人没有首先发现）时，才会发现出了问题。

当你拥有一个 API 的代码时，相反的问题会产生。你的 API 可能提供有价值的服务给客户。它可能可以访问所有的机密数据，其中一些是客户应该要访问的，还有一些则应该隐藏起来。错误的输入可能造成你的 API 以意料之外的方式作出反应，通常会导致对 API、数据以及服务的破坏。因此，你需要找到净化输入数据的方法，但也要在这个净化过程不能按预期奏效时提供保护。

无论你的 API 在一个请求中是发送还是接收数据，在沙盒或虚拟环境中使用该 API 都是有意义的，因为这样做会降低风险。完全隔绝坏的东西是做不到的，但你可以将风险以及你所遇到的漏洞带来的影响降到最低。接下来会构建一个在沙盒或虚拟环境中使用 API 的例子。

10.2.1 确保API可以工作

安全有关风险管理。确保你的 API 按预期工作可以降低风险，因为这会减少别人以你未考虑到的方式运行 API 的可能性。黑客常常干着寻找方法使代码异常工作这样的肮脏事情，让代码执行其作者不想其做的事情。使用 API 安全区域可以让你用平时无法使用的测试方法去测试代码，而无需在测试服务器上制造问题。你可以执行以下测试来确保 API 按预期工作。

- 验证正确的输入会产生正确的响应
- 运用范围检查来确保 API 能正确地响应边界外的数据
- 输入错误类型的数据来确保 API 能根据类型净化数据
- 专门输入黑客可能使用的非法数据，比如脚本、标签或二进制数据
- 删减必需的数据
- 加入额外的数据
- 制造 null 值的输入
- 使 API 超负荷
- 使 API 得不到资源

10.2.2 实现快速开发

把 API 直接放在服务器上意味着你提供了 API 的一份副本给每个人使用。当然，一旦任何一个开发人员决定尝试某些激进的做法而导致 API 或服务器崩溃，其他所有人也不得不停止工作。你应该让开发人员在做试验时没有负担，因为黑客在做这些事时肯定不会有负担。创建一个所有人都必须遵守规则的受约束环境，几乎肯定会把那些黑客用来破坏服务器的安全漏洞隐藏起来。

沙盒通过使服务器变得不太可能崩溃来帮助实现快速开发。如果 API 遇到了问题，你通常可以非常快速地恢复它。沙盒环境还能很容易地模拟现实世界中的各种情况，比如缺少资源或完全没有资源的情况。但是，你仍然要有一个 API 的副本在运行，所以还不能像在开发环境中那样轻松。

使用虚拟环境可让每个开发人员（或者每组开发人员）拥有一份 API 的副本。当然，你要使用更多的服务器资源去实现这一场景，但关键是你最终会有一个不会干扰其他开发人员的编码工作的环境。每个人都与其他人隔离。如果一名开发人员想到一个攻击 API 的好点子并且攻击成功了（使得 API 或虚拟环境崩溃），这些工作不会影响到其他人。这里的要点是要提供一个环境，让开发人员在寻找最好的应用解决方案时可以没有负担地去做任何事情。

这个例子中要考虑的权衡点是，资源与灵活性要选择哪一个。沙盒方法会对资源比较友好，这对于那些没有太多硬件资源的小公司可能是一个较好的选择。虚拟环境对于正在使用需要很多开发人员关注的较大型 API 的公司是一个比较好的选择。使开发人员不会影响其他人是这个例子中首要考虑的因素，且额外硬件的使用可能不是一个太大的问题。

当要获取硬件来进行开发或测试时要有些突破性的思维。那些不再能正常提供功能的系统可能仍然有足够长的寿命可用于开发目的，特别是如果你的目标是在一个受约束的环境中进行测试。总之，在搭建一个测试环境时不要总想着用新的硬件，有时候在进行设备升级之后，旧的硬件也能工作得很好，并且你手边可能就有很多这样的资源。

10.2.3 证明最佳的集成

许多集成测试存在的一个问题是测试在找正向的结果而不是负面的结果。开发人员希望验证以下事项。

- API 按说明书所说的那样工作
- API 的功能匹配该 API 必须完成的任何业务逻辑

可用来进行集成测试的方法有很多，它们还会帮你找到潜在的安全问题。下面描述了最常见的技术：沙盒、创建虚拟环境、使用 mocking 以及依靠 API 虚拟化。

1. 沙盒集成测试

使用沙盒环境可以执行逼真的测试。在某些情况下，当沙盒环境在某些方面不能匹配现实环境时，你会得到假阳性的测试结果。此外，使用沙盒环境通常不能体现出 API 真正的速度指标，因为在 API 和应用程序之间有另一层软件。当多名开发人员在不符合生产环境的环境中发起请求时，速度问题会被放大。

如果你发现你进行的测试得出了不可信或不一致的结果，那么就需要创建一个更好的测试环境。当输入特定的值无法获得特定的输出时，测试过程可能会得出假阳性测试结果，让你花费很多天去解决这个问题。假阳性在处理安全问题时是特别令人头痛的。你可能在某一时刻有可以产生问题的条件，而下一刻又没有了。

2. 虚拟化集成测试

另一种替代方案，虚拟环境，同样提供了逼真的测试。由于虚拟环境的工作方式，它可以更大程度地模拟生产环境，所以你不会得到太多的假阳性结果。实际测试的速度通常不是一个问题，因为每个开发人员都有自己的私有环境。但是，除非你有正确的配置，否则就不能可靠地进行应用程序的速度测试或负载测试，因为虚拟环境会有偏差的结果。当使用虚拟环境时，在集成测试发生一次失败之后的恢复速度肯定比较快。

3. 使用mocking进行集成测试

一些开发人员使用 mocking 对 API 进行集成测试。一个模拟的 API 以脚本的方式接收输入并提供输出。换言之，这个 API 不是真的。模拟 API 的目的是在分离 API 的情况下测试应用程序部分。使用模拟的 API 可以在开发团队开发 API 之前就开始构建和测试应用程序。

mocking 的好处是任何测试都能非常快地运行起来。此外，应用程序接收到的响应是精确且不会变的，不像一个真正的 API，其响应可能会变化。（mock 不会执行任何真实的执行过程，所以不会有变化的响应。）但是，mock 里面的错误会在测试应用程序时制造假阳性的结果。

在网上就有模拟 API 的例子。例如，你想要测试基于 REST 的应用程序，就可以使用 Mocky（http://www.mocky.io/）来执行这个任务。你可以使用 Mocky 为应用程序创建一个可预期的响应，如图 10-1 所示。Mocky 还可以创建各种响应类型。你不仅控制了 HTTP 响应，还控制了内容的响应类型和值。

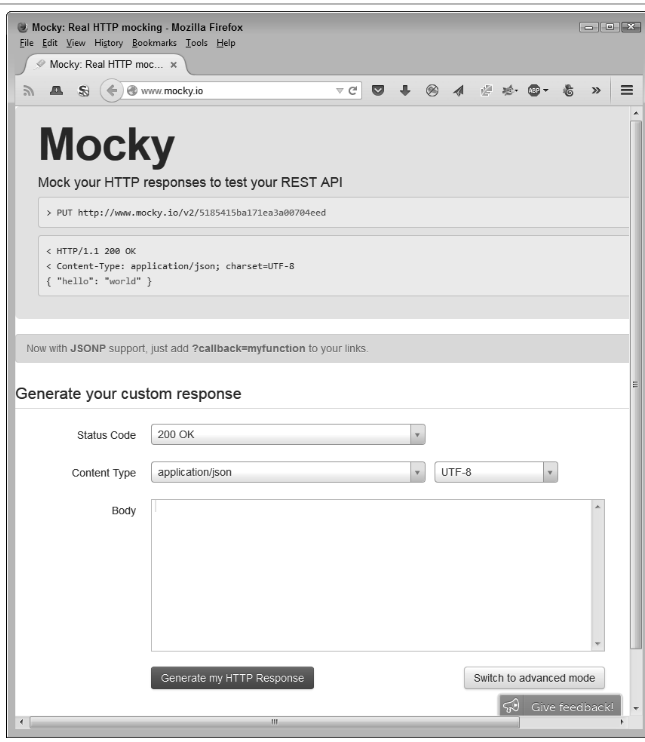


图 10-1：Mocky 提供了一个供你填写的简单表单来配置测试

许多 mock 提供了 SOAP 和 REST 支持。例如，mockable.io/（https://www.mockable.io/）同时支持 SOAP 和 REST，如图 10-2 所示。要使用这个服务，你必须注册一个账户。一个试用账户就能让你执行测试，但你创建的任何数据都不是私有的。为了创建一个私有配置，你必须获得一个付费账户。

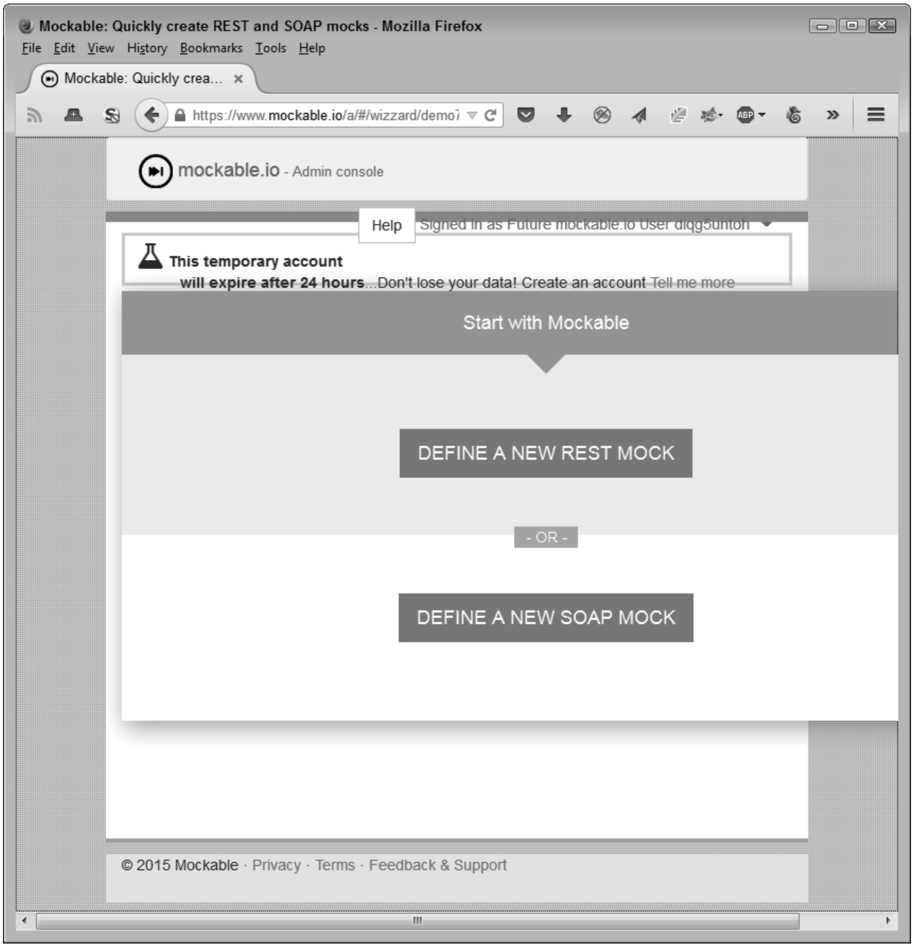


图 10-2：当你需要 SAP 支持时可以使用 mockable.io 等 mock

在这个例子中，一旦你选择了想要创建的 mock 类型，就填写表单去执行真正的测试过程。如图 10-3 所示，你可以更改的 mock 的配置是很广泛的。在这种情况下，你还可以选择诸如用于创建调用的 verb 等项目。关键是 mock 提供了一个环境，使你可以用某种特定方式提供特定的输入并得到特定的结果。

更高级的 mock，比如 Apiary（https://apiary.io/），会提供许多服务来使 mocking 的过程更加简单。你可以使用这个服务来创建文档、集成代码样例、执行调试任务以及自动测试。当然，随着 mock 的复杂程度的增加，价格也会增加。

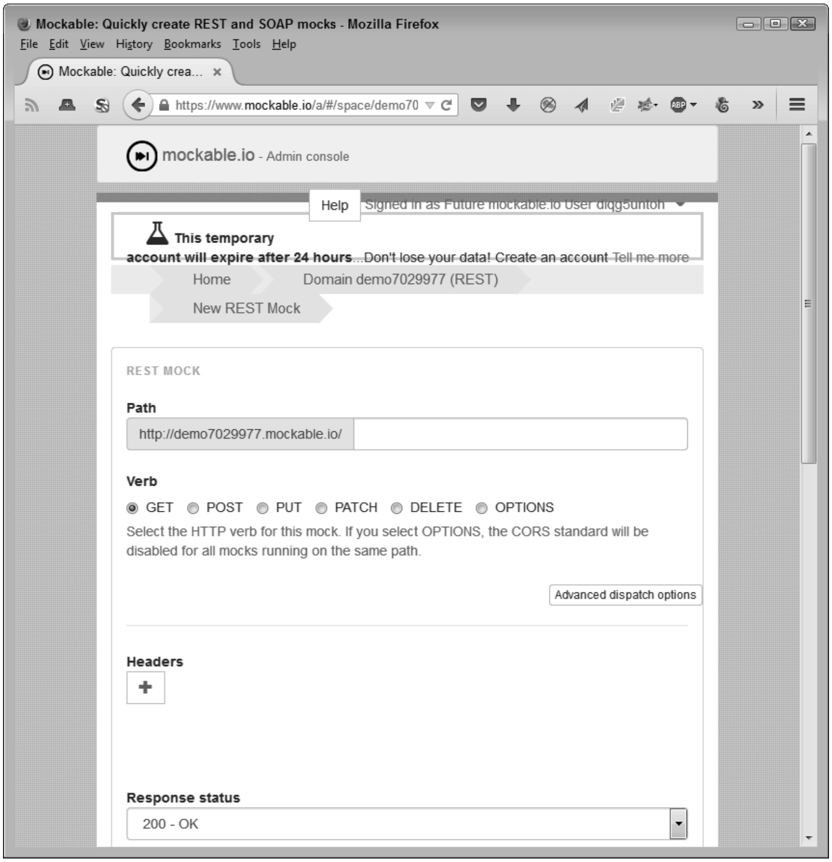


图 10-3：某些 mock 对测试界面提供了大量的控制

4. 使用API虚拟化进行集成

沙盒环境可以做一些你平时不容易做到的事情，比如虚拟化真实的 API。很有可能你的开发排期与你所依赖的某些第三方服务的排期不一致。当发生这个问题时，公司常常只能等待所依赖的 API。当然，你可以重新开发自己的 API，但大部分公司不会这样做，并且这真的不是一个好的选项，即使你有相关资源。

永远不要混淆 API 虚拟化和服务器虚拟化或虚拟机的使用。在后面两种情况中，你会创建一个完整的虚拟系统，模拟一个标准平台的每个部分。API 虚拟化只是模拟 API。你不能用它来执行如直接运行一个应用程序这样的任务。

虚拟化是一个创建 API 表现的过程，可用于测试或其他目的。这个表现作为一种黑盒，可用来替代真正要依靠的真实 API。这个黑盒最终会提供对真正 API 的访问，但也可以提供对作为替代者的 mock 的访问。关键是 API 虚拟化层提供了与 API 交互和集成到应用程序的一致方法，即使是在 API 还没有完成的情况下。图 10-4 展示了 API 虚拟化层的样子。

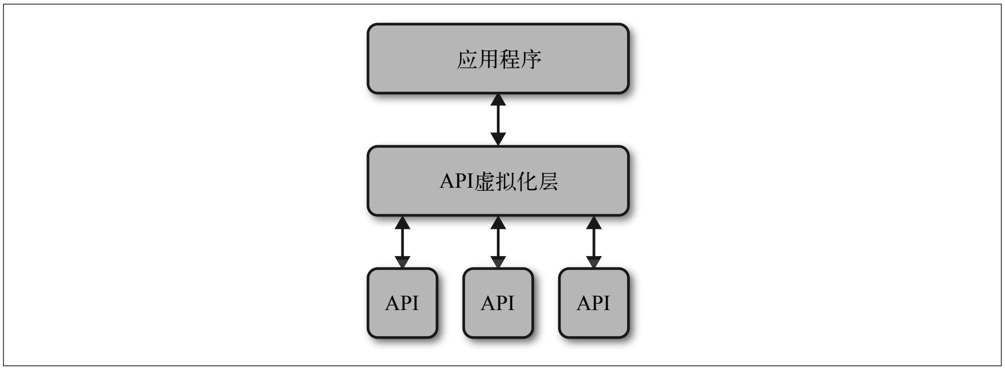


图 10-4：API 虚拟化层使应用程序与 API 相分离

使用 API 虚拟化可以使你的应用程序经受住各种失败。API 虚拟化层可以让应用程序即使在 API 不能正常工作时继续运行。例如，当错误发生时，你可以创建一个 mock 来替代 API，从而使应用程序能继续运行。用户甚至可能不会感到任何不同。当 API 恢复正常工作时，你可以在后台执行任何所需的修复。这里的关键点是黑客不能简单地通过增加 API 的负载来使应用程序遭遇失败。

在某些情况下，你可以结合沙盒产品和 API 虚拟化来创建一个更好的环境。例如，SmartBear 提供了两款产品：VirtServer 和 Ready API!（https://smartbear.com/product/ready-api/overview/），可以创建一个两层方法来使用 API。（请参阅 http://smartbear.com/product/ready-api/servicev/features/share-virtual-services/，获取更多细节。）其要点是在处理复杂的 API 交互时获取额外的灵活性。

10.2.4 在负载情况下验证API的表现

负载测试对于安全来说很重要，因为黑客常常使 API 或应用程序过载来导致其以某种方式失败。在 API 或应用程序失败之后，黑客会使用各种途径来利用失效的软件作为入侵网络的入口。因此，知道当负载加重时你的 API 会如何退化是很重要的，因为你想要优雅地失败，并且是以一种不会给黑客打开很大入侵漏洞的方式。重要的是要记住，如果你给了足够大的负载，每个 API 都会在某个点发生失败。尽管你可能在过去看得了一些承诺，但没有任何方法可以让软件伸缩到某个点从而使其可以承受无限的负载（假设有可能可以进行这样的测试）。

为了执行负载测试，你必须能够模拟负载，这通常意味着要给发起调用请求的系统增加软件，减少所需的资源并进行任何所需的日志记录。大部分开发人员会提供一个沙盒环境来进行负载测试，因为沙盒环境提供了最真实的测试环境。寻找一些能用在你需要支持的所有平台上的工具。某些产品，比如 LoadUI（http://www.loadui.org/），提供了用于 Windows、Mac 和 Linux 测试的版本。在获得任何工具之前，先尝试下载试用版本来看看其是否能如宣传的那样工作。

任何 API 负载测试的起点都是确定如何测试。你需要获取统计数据来了解如何加载 API 以模拟一个真实的环境。为了确保 API 如宣传的那样工作，你需要确定以下各项。

- API 每秒接收的平均请求数
- API 每秒接收到的请求峰值
- 端点（发起调用 API 的地点）的吞吐量分布
- 用户或工作组的吞吐量分布

确定如何创建测试 API 的连接也是很重要的。你可能会很随便地就开始测试。但是，很重要的是要采用各种类型的连接来测试 API，以确保其能在所有的环境下表现良好。

- 制造重复性负载（测试软件顺序使用相同的请求）
- 模拟的流量模式（一个随机版本的重复性负载或一段 API 访问日志数据的重播）
- 真实的流量（你有一个测试组用真实的请求访问 API）

在建立起能在标准条件下运行的 API 之后，你需要改变这些条件来模拟黑客的攻击。例如，黑客会用特定的请求大量攻击 API，希望使其以某种方式失败。黑客还会在一次 DDoS 攻击中持续增加越来越多的僵尸请求，从而能很简单地让 API 到达被破坏的边缘。这真是有创意。很重要的是要确定什么会破坏 API 并导致其失败，然后看看 API 会如何失败。知道 API 会如何失败将有助于你确定风险级别，这些风险每个黑客都会感兴趣，这样你就能更好地为后果做准备。

10.2.5 使API远离黑客

在执行了本章描述的所有其他测试之后，你可能会认为你的 API 是安全的。问题是你并没有考虑到所有黑客会采用的攻击方式。如果某人真的决定要破坏你的 API，他们就会找到破坏的方法。第 9 章讨论了需要像黑客一样思考。当然，那是一个好的开始，但一旦你想到某人能用来破坏你的 API 的一些方法，就需要测试它们。你能安全地进行测试的唯一方式是创建一个 API 安全区域。

在这种情况下，使用沙盒和虚拟环境测试会帮你得到最好的结果。任一种环境都能为测试者提供特定的好处，来帮忙确保 API 在生产环境中能表现良好。

> 在生产环境中测试你的应用程序及其相关的 API 永远不是一个好主意，除非有绝对的必要。许多开发人员都尝试过这样做，并发现测试过程确实会损害应用程序应该保护的数据。此外，你通常不能安排出方便用户的测试计划。因为用户在测试环境中增加了变数，所以你需要在用户不大量执行任务时进行测试。否则你无法确定一次失败是测试还是用户输入造成的结果。

10.3 用API沙盒进行开发

虚拟化一个 API 沙盒是很容易的。这就与给孩子做一个沙盒一样。API 可以访问沙盒中的任何东西，而不能访问外部的任何东西。将 API 保持在一个沙盒里有很重要的安全意义，因为它使得 API 即使在遭到黑客攻击时，也不太可能对应用程序或其数据造成损害。通过保持 API 的隔离，它只能损害到已经分配给它的资源。这意味着 API 绝对不可能以不受控制的方式造成损害，除非设计者没有正确或有效地设计沙盒。

没有什么是免费的。把 API 放在沙盒中也很可能会让 API 不能按预期工作，或在资源紧张的环境中工作得很慢。关键是要创建使 API 的功能保持可靠的 API 沙盒。创建一个平稳的 API 沙盒是很难的，这需要对 API 如何与应用程序一起工作来执行特定任务有深度的理解。（不需要理解 API 的每个细节，因为应用程序不太可能使用整个 API。）

许多文章对沙盒小题大做，但事实上，它与使用任何现代操作系统没有什么不同。从前，应用程序能够完全访问整个机器。现在，一个应用程序控制了对系统资源的访问，包括内存和处理时间。应用程序必须与在同一系统中运行的其他应用程序共享资源，所以操作系统控制着对资源的访问。使用沙盒，你可以有一个额外的控制层，但这个控制在功能和效果上与任何其他的应用控制类似。

一旦你有开发一个 API 沙盒的良好理由，就应该开发一个沙盒用于你的应用程序。事实上，对于应用程序中的每个 API，你都需要一个独立的沙盒。某些应用程序使用了相当多的沙盒，而这可让一个 API 不被另一个 API 产生的问题所污染。保持每件事情是隔离的且不能破坏其他的事情通常是一个好主意。在某些情况下，你可能不得不在构建应用程序的方式上变得有创造力，使得 API 可以持续正常运行，但使 API 保持在完全隔离和受控的环境中是一个很好的想法。

沙盒的开端

你可能会认为沙盒是相对较新的概念。但是，从 20 世纪 70 年代起，沙盒的概念就已经以 Hydra 操作系统的形式存在了很长一段时间。卡耐基 · 梅隆针对迷你计算机的多处理器操作系统有一个沙盒功能，在其中开发人员能够以相对安全的方式试验人工智能（AI）应用（http://research.microsoft.com/en-us/um/people/gbell/Computer_Structures_Principles_and_Examples/csp0366.htm）。每个应用程序作为用户应用运行，这意味着它不能访问较低层的、可以导致系统失败的操作系统功能。

其他一些开发人员注意到了 Hydra 提供的功能并在其他环境中重新实现了它。例如，Sun 公司提出了动态系统域（Dynamic System Domains，http://www.filibeto.org/~aduritz/truetrue/e10000/dynamic-sysdomains.html）的概念来提供错误保护和应用隔离（包含在其他目标之中）。FreeBSD 实际上会通过将应用程序投入 jails（https://www.freebsd.org/doc/handbook/jails.html）的方式来锁住它们。因此，沙盒的概念由来已久，但是将其用于 API 则是相对较新的理念。

现在，许多应用程序在沙盒中运行。例如，2012 年，Apple 开始要求开发人员在 Mac 应用商店中的应用程序中包含沙盒特性（https://developer.apple.com/app-sandboxing/）。Google 的 Chrome 浏览器（https://tools.google.com/dlpage/res/chrome/en-GB/more/security.html）、Microsoft 的 IE（http://securityintelligence.com/internet-explorer-ie-10-enhanced-protected-mode-epm-sandbox-research/）以及 Apple 的 Safari（http://www.cnet.com/news/safari-matches-rivals-with-sandboxed-flash-for-better-security/）都有沙盒。唯一没有沙盒的浏览器是 Mozilla 的 Firefox（http://www.extremetech.com/computing/178587-firefox-is-still-the-least-secure-web-browser-falls-to-four-zero-day-exploits-at-pwn2own）。

10.3.1　使用现成的解决方案

构建自己的沙盒会有趣，但更有可能是在浪费时间。你应该聚焦于自己的应用程序，而不是再去构建一个能安全使用它的环境。这就是为什么你需要一个你能信任的现成解决方案。下面列出了许多可用的沙盒解决方案。

- AirGap（https://spikes.com/isolation-is-the-new-prevention.html）
  这是一个有趣的想法。不是让浏览器在客户端系统中运行，而是让其在云端的服务器上运行。用户依赖某个查看器应用程序，该应用程序提供了与浏览器的隔离。你会看到浏览器窗口并像平时一样与其交互，但唯一在客户端系统中运行的是浏览程序。供应商用这个方法承诺物理、连接、会话和恶意软件的隔离。你需要用你的应用程序来尝试一下，看这种方法是否能真正完成工作。

> 对于大部分用户来说，一个像 AirGap 这样的方案能够工作得很好。但是，你必须考虑浏览器在哪里运行。如果你的公司属于受监管行业或者你的应用程序在处理敏感信息，那么 AirGap 就不适合你。在这种情况下，浏览器确实应该在宿主系统上运行以正确地工作并维持机密性，这是法规（或常识）所要求的。

- Sandboxie（http://www.sandboxie.com/）
  你可以使用 Sandboxie 来在宿主系统上运行任何应用程序，而不只是浏览器。这意味着你能用 Sandboxie 来执行各种其他解决方案可能无法支持的任务。因为它在客户端系统上运行，所以你的敏感数据会得到完全的保护。供应商提供了一个你可以测试的试用版本。当你要购买一个授权许可时，可以选择家用版、小型商业版或者企业版。只有企业版有支持包。Sandboxie 只能在 Windows 操作系统上工作。

- Spoon.net（https://turbo.net/browsers#spoon.net）
  有时候你需要测试大量不同的浏览器，但将它们全都下载并安装到你的系统上是很困难的。根据你的需要配置所有这些浏览器也会增加额外的工作。通过 Spoon.net 提供的环境访问浏览器可让你避免这些痛苦。要使用这个产品，你要在浏览器上安装一个插件，然后选择你想使用的浏览器，如图 10-5 所示。浏览器的选择不受限于你的平台所支持的浏览器。例如，尽管你正在使用 Windows 系统，也可以加载 Safari 或 Firefox Mobile 来执行所需的测试。

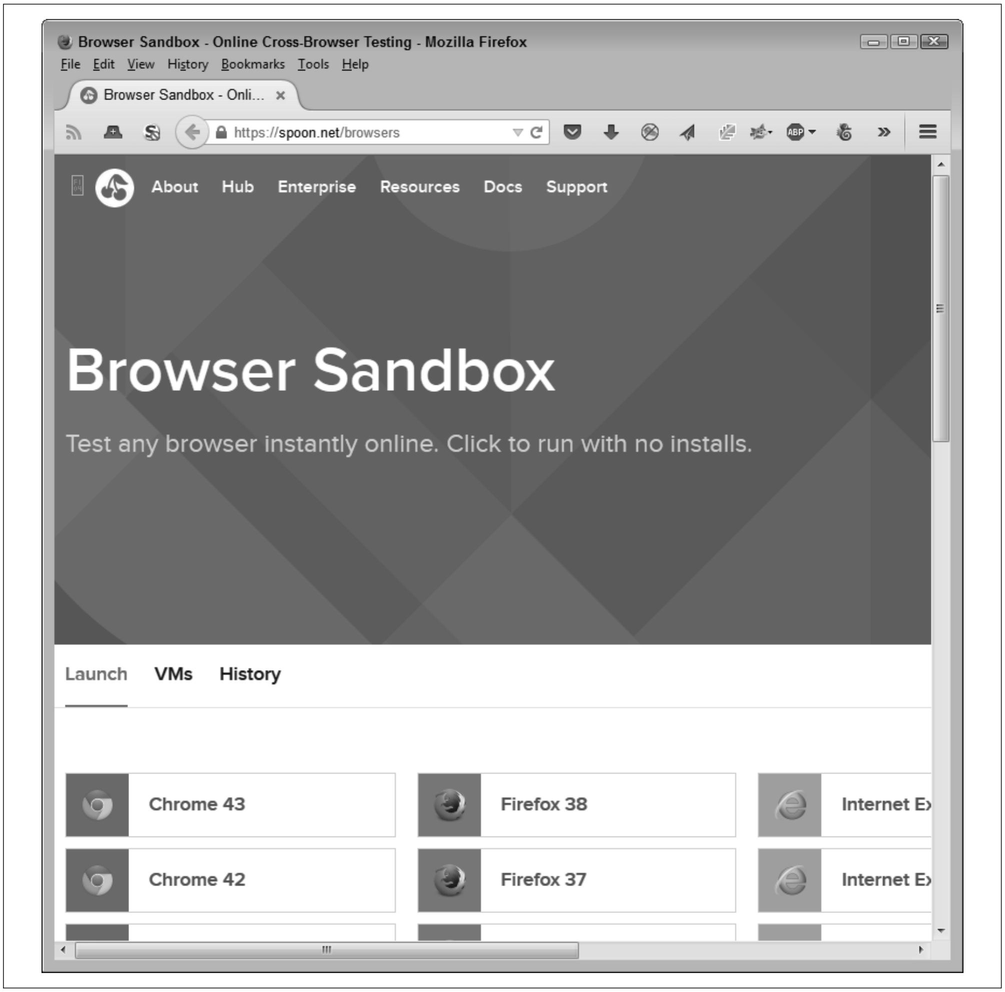


图 10-5：Spoon.net 可以使用所有常见的浏览器

10.3.2 使用其他供应商的沙盒

有些情况下，无论你是否想要，都得使用沙盒。例如，当开发一个使用 eBay API 的应用程序时，你必须首先使用沙盒（https://go.developer.ebay.com/developer-sandbox）来编写应用代码并测试它。只有在 eBay 验证了应用程序之后你才能在生产环境中运行。由于 API 环境所涉及的高风险，你可能会发现你要花更多的时间来使用第三方沙盒测试你的代码。

访问大部分这些沙盒要依靠专门的密钥。供应商，比如 eBay，想要知道谁为了什么目的正在使用他们的沙盒。这个密钥唯一地标识你的身份，而你必须拥有它来调用沙盒。当使用 eBay 的时候，获取密钥是很容易的，注册一个账户，然后进入沙盒开始执行应用测试，如图 10-6 所示。

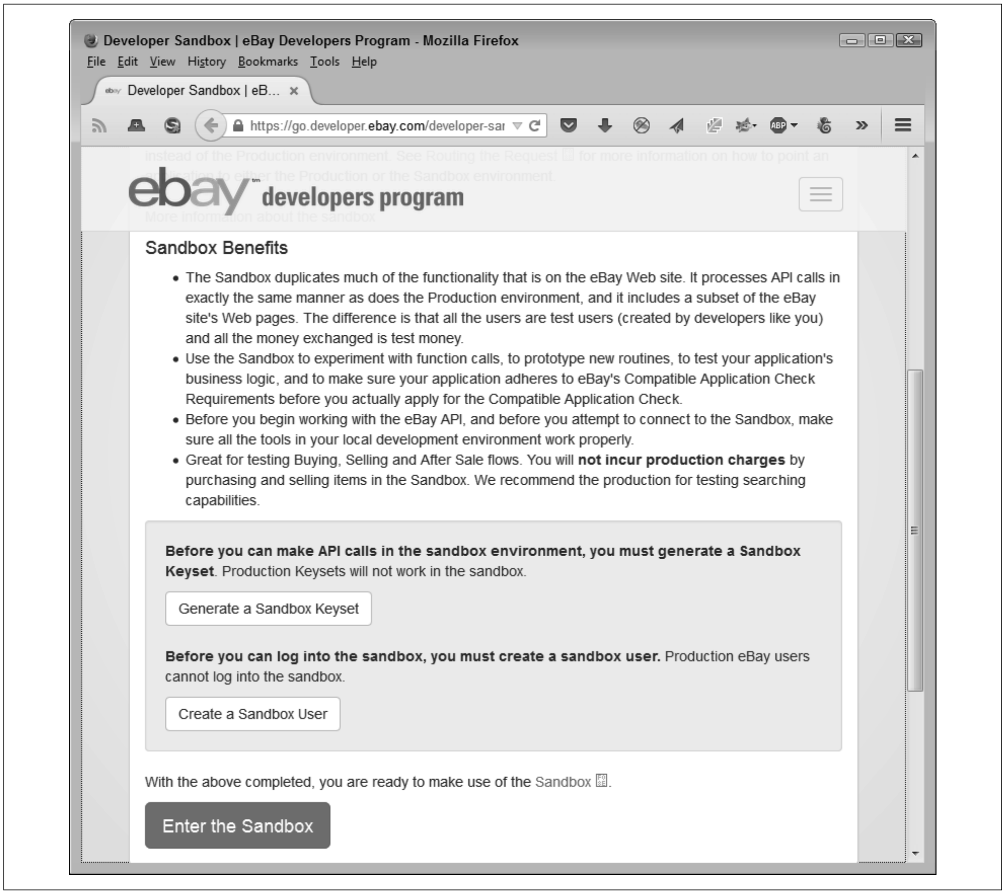


图 10-6：大部分供应商会要求某些访问控制以使用他们的沙盒

当你在应用程序中需要使用多个 API 时会有些问题，每个 API 都要有自己的沙盒环境。尝试让应用程序获得认证意味着要让每个供应商都签署认证。请确保你计划在开始编码前让应用程序获得每个供应商签署的认证。在某些情况下，为了获取需要的验证，你可能不得不将应用程序分解为几个可测试的部分。

> 在单个应用程序中使用多个 API 已经变得很普遍，而且很复杂，这让许多开发人员转为向 API 集成求助。使用 API 集成可以通过创建 API 整合来降低复杂度。使用 API 集成超出了本书的范围，但你可以在“Api Aggregation: Why It Matters and Eight Different Models”（http://www.programmableweb.com/news/api-aggregation-why-it-matters-and-eight-different-models/2013/12/13）和“Extending REST APIs with API Aggregator”（http://tech.3scale.net/2013/04/18/accelerate-your-mobile-api-with-nginx-and-lua/）这两篇文章中找到很多有用的相关信息。当然，还有很多其他资源可作为集成化的参考，但这两篇文章可以作为一个很好的开始。

10.4 考虑虚拟环境

有时候你的 API 需要在一个虚拟环境中工作。一个虚拟环境可以给人真实环境的感觉，但不会让 API 导致问题。通过本章，你已经看到了沙盒和虚拟环境的比较。从安全的角度来说，这两者都能对抗黑客的行为，但是，是以完全不同的方式。然而，最重要的一点是分离 API、应用程序、应用程序组件或任何其他你想要与操作系统隔绝的软件项。隔绝可以让你拦截和清理数据，以及保持软件处于受控状态，这样黑客造成的伤害就会较小。下面将详细讨论虚拟环境。

10.4.1 定义虚拟环境

虚拟环境可以指代任何提供隔绝其他软件方法的软件。你可以将其想为一个容器，因为它就是。本章前几节已经描述了各种虚拟环境。但是，这一节会描述用于开发目的的那种虚拟环境，而不是作为用户来运行应用程序。

采用虚拟化的基本理由是要隔绝所有或部分开发环境，这样你就能任意操作应用程序。发现安全问题实际意味着你要用在标准的操作系统中永远不会使用的方式去体验软件（特别是没有人会连接到你所使用的网络，这样你的错误就不会影响到任何人）。

有些语言有自己的虚拟环境工具。例如，Python 用户有很多创建虚拟环境的工具。这些工具使得为特定语言搭建特定环境变得更简单。你可以在网页 http://docs.pythonguide.org/en/latest/dev/virtualenvs/ 上了解 Python 所能提供的服务。其要点是查找可能的针对具体语言的工具来创建可用于该语言的虚拟环境。

虚拟环境还能包含比沙盒更多的特性。可以创建带有特定操作系统、开发工具、应用搭建和浏览器的虚拟环境。虚拟环境可以模拟开发过程中的每个部分，这样你就能在一个虚拟环境中安装 Apache 服务器而在另一个环境中安装 IIS 服务器。这两个虚拟环境可以存在于同一台机器上而不会产生冲突。

10.4.2 区分虚拟环境和沙盒

虚拟环境和沙盒有许多相似点。本章前面描述了二者最常见的相似点，比如保证 API 安全。大部分虚拟环境和沙盒的基本区别在于虚拟环境具有以下特征。

- *可重复*
  当你将虚拟环境文件迁移到另一个系统时，那个系统可以像原来的系统一样正常运行虚拟环境。这意味着每个开发人员都可以以相同的方式看到应用程序，使用着相同的开发环境，所以在一个系统中看到的结果可以在另一个系统中重现。环境搭建时的不同会导致难以（有时候是不可能）验证安全问题的出现。

- *可移动*
  根据你所使用的虚拟化软件，你可以将虚拟环境移动到任何地方。只要接收到虚拟环境文件的开发人员有正确的软件，就可以精确地重建开发项目所需的虚拟环境。

- 可恢复

  虚拟开发环境依赖一种特殊的文件，它定义了如何创建所需的仿真环境。如果你的试验导致了虚拟环境崩溃，你所需要做的就是关闭虚拟环境的这个副本并启用另一个。恢复只会花费软件加载所需的时间，可能只需要几秒钟。

- 可重建

  有时候你需要在你的系统上运行同一个项目的多个副本。你可以创建你所需要的任意多的虚拟环境副本。虚拟环境的特殊属性意味着你创建的每一个副本一开始都有完全相同的资源、加载的软件、环境等。因此，你系统中的每一个副本都是其他副本的复制品。

10.4.3 实现虚拟化

一个最有趣的虚拟开发环境是 Vagrant（https://www.vagrantup.com/）。这个产品可以在任何 Windows、OS X 或常用的 Linux 系统（有很多版本的 Linux 不能使用 Vagrant，所以一定要检查列表以确定 Vagrant 支持你的 Linux 发行版）上运行。要使用这个产品，要在系统中安装它，配置你想用于开发的环境，并把你的项目代码放到结果文件中。任何时候你想要启用这个项目的开发环境，只需要输入 `vagrant up` 命令，它就会为你配置正确的环境。无论你把项目文件移动到哪里，这个产品都会创建出相同的环境。Vagrant 对于独立的开发人员或小型公司是一个更佳的选择，它不贵且易于使用。

另一个有用的虚拟环境产品是 Puppet（https://puppetlabs.com/）。这个产品依赖 VMware（http://www.vmware.com/），Puppet 负责执行管理任务。Puppet 与 VMware 的结合能更好地适应企业环境，因为 VMware 有大量的工具可供你使用，并且它还能提供对大型业务的支持。

10.4.4 依靠应用程序虚拟化

目前存在各种各样的虚拟环境以满足你能想到的任何需求。例如，Cameo（http://www.cameyo.com/）可以将任何应用程序打包成单一的可执行文件（.exe 文件），然后你可以将它复制到任何的 Windows 计算机且不需要任何安装就能运行它。如果你需要支持其他平台，可以将应用程序复制到 Cameo 云服务器上并在那里执行它。

应用程序虚拟化是一种打包封装的方案。你要创建一个专门满足这个应用程序的虚拟环境。图 10-7 展示了虚拟应用环境可能是什么样子。就应用程序而言，它正在其原生的环境中执行操作，尽管它正在使用一个完全不同的操作系统。

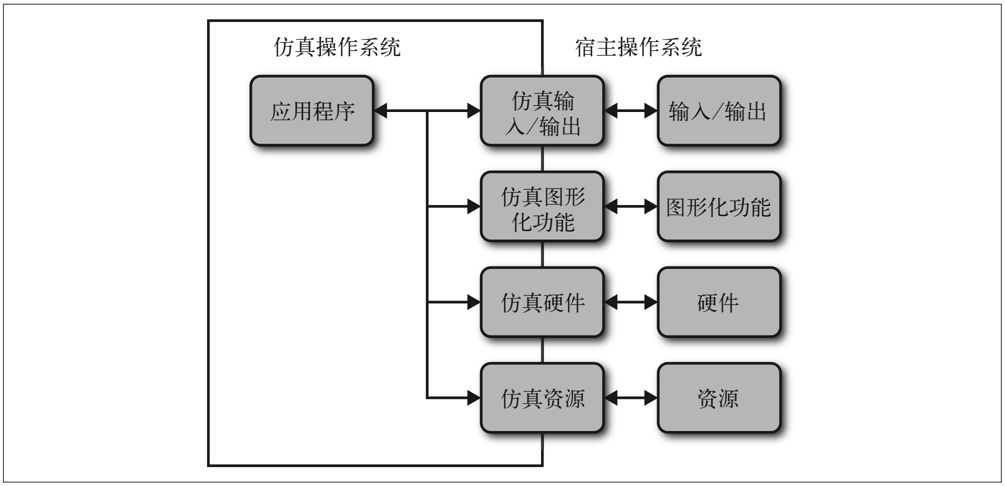


图 10-7：应用虚拟化是一种打包技术

另一个能提供应用程序虚拟环境的产品是 Evalaze（http://www.evalaze.de/en/evalaze-oxid/）。这个产品强调的是它连同应用程序虚拟化一起提供的沙盒环境。你可以用它来使应用程序在各种地方运行，包括在记忆棒中。如果你的应用程序要求一个特定的浏览器，并且你想要确定用户的环境是正确配置的，那么这是一个很好的解决方案。

研究一个合适的虚拟化方案会花费时间，并且你需要以开放的心态来进行研究。某些看起来怪异的解决方案可能正是你为自己的应用程序创建安全环境所必需的。

第 11 章

检查库和API的漏洞

在了解某个软件以怎样的方式运作之前，你需要对它进行测试。是的，你有一份文档并且会依据文档来编写代码。你甚至会调试代码来确保没有明显的错误。但是，在进行某种级别的测试之前，你无法知道这些代码是否实现了最初想要实现的功能。测试能够确保软件像你期望的那样运作。

本章会讨论传统意义上的测试，但也会涉及非传统的测试类型。一个标准的测试框架会提供特定的输入，然后验证期望的输出。但是，现实世界并不总是会提供这种库和 API 期望的输入，所以测试其他输入也是很重要的。当然，以这种方式测试时，你不知道要期望什么作为输出。库或 API 的设计必须包含能从错误输入中恢复的功能，这样系统才不会崩溃。但是在执行所需级别的测试之前你并不知道这个功能是否存在。

测试能够并且应该在多个级别上进行。单元测试，即对单个代码块的测试要首先进行。开发人员要在编写完代码后不久就开始执行这类测试。接着要把应用程序各部分组合在一起进行集成测试。最后，应用程序作为一个整体与所有已完成的软件进行测试。所有这些级别的测试（以及更多测试）都要求进行安全测试，即检查意料之外的输入和确定代码是否以可接受的方式执行。在此期间，开发人员还要创建测试框架，即能自动进行测试过程的测试程序，这样测试就能更加可靠和一致。

每种开发语言都有开发人员必须检查的特定问题。本章的最后一节会探讨在执行安全测试时如何检查具体的编程语言问题。与其他测试领域不同，特定语言的问题会围绕着安全进行测试，因为黑客常常寻找这些不同点来作为攻击方法，通过其可导致应用程序以特定方式崩溃或使应用程序出现逻辑错误，这样黑客就能利用其进行入侵。跟踪语言的缺陷与跟踪第三方库和 API 的缺陷一样重要。你需要以各种可能的角度来测试应用程序的每个部分，以确保应用程序会以预期的方式运行。

跳出常规思维的测试

代码依赖过程。是的，代码可能是事件驱动的，或者它可能不包含状态的概念，但本质上，代码是依赖一系列步骤来完成任务的。这一系列步骤就是一个过程。在这里，过程是否完成无关紧要。

我上小学的时候第一次见识了跳出常规思维的测试。那段时间的大部分事情已经不记得了，但有一件事在这些年里一直作为我的一项生活技能而被我牢牢记住。老师让我们所有人写一个烘烤面包的过程。这看起来是一个很简单的要求，但是我们所有人都失败了。每个人想的都是，把一片面包拿出来并放入到烘烤机中。等待烘烤的面包弹出来没有问题，之后给烘烤的面包涂上黄油也没有问题。

问题出在了将每片面包从包装纸中拿出来的环节。当老师按照我们写的过程操作时，是的，她尝试将每片面包连同包装纸一起放入了烤箱。我们所有人都做了一个有缺陷的假设。计算机就像我的老师。我们基于常识和经验来假设计算机将会做的事情，但这两样（常识和经验）计算机都没有。跳出常规思维的测试意味着，在开发代码的过程中每个人都能做出每片面包都在包装纸里的假设。安全漏洞常常是由于开发人员不应该做的假设引起的。

11.1　创建测试计划

每个测试场景都需要一个测试计划。虽然你想要有跳出常规思维的测试，但确实需要一些结构化的方法来执行测试。否则，测试会变得不一致和不完整。为了达到有用的目的，测试需要有条理，并且有足够的灵活性能够在你需要的时候提供更多测试的能力。为此，下面将帮你从开发的角度制订一个测试计划。

> 要特别注意的是，测试计划常常有很多阶段和方向。应用开发工作中的每一位利益相关者都会想执行某些级别的测试，以确保应用程序满足特定的目的和目标。这些观点可能会发生冲突，但大部分情况下，它们只是相互补充。例如，DBA 可能想验证应用程序与数据库的交互方式是与公司的指导原则相一致的。

11.1.1　考虑目的和目标

除非开发团队定义了测试的目的（goal）和目标（objective），否则测试不会成功。简单来说，目的就是确定应用程序是否能满足公司要求的技术和业务需求。而目标则是确定应用程序在某个业务环境中能够成功地执行一些任务。例如，某个目标可能是要将新用户添加到数据库而不会产生错误、重复的用户或泄露核心信息。下面将讨论为了得到一个可用的结构，测试必须满足的目的和目标。

制订测试的替代方案

测试不是检查应用程序的唯一方法。测试的要点是验证应用程序以确定的方式运行。其他技术也能达到同样的目的。测试最常见的替代方案是代码检查、统计分析、模型检查和论证。每一种方案都可作为你提升应用程序安全性和确保应用程序可靠运行的策略的一部分。

代码检查是人工进行的检查。一个团队会审查一遍代码并验证其设计和实现满足规范的要求。采用代码检查有助于定位自动化方法可能遗漏的潜在编码问题。事实上，应用程序很有可能带着严重的缺陷通过编译并执行，而代码检查可以发现这些缺陷。你可以在网页 https://www.owasp.org/index.php/Code_Review_Introduction 上查看更多关于代码检查的信息。

统计分析的过程与代码检查一样，但它使用了自动化工具而不是人工进行检查。统计分析的优点是一致性和速度。自动化工具会以完全相同的方式检查应用程序的每一个方面，并且不会由于疲劳而产生错误。此外，自动化工具比人工作得更快。但是，自动化工具也可能遗漏那些人工几乎可以立刻发现的错误。你可以在网页 https://www.owasp.org/index.php/Static_Code_Analysis 上查看更多关于统计分析的信息。

模型测试会验证应用程序的属性是否满足规范的要求。大部分情况下，这意味着要验证应用程序是否能提供算法等要素的解决方案。此类测试是自动化的，但需求大量的人工输入才能执行。你可以在网页 https://www7.in.tum.de/um/25/target.html 上查看更多关于模型测试的内容。

论证常常表现为对应用程序进行某些形式的压力测试。论证要验证应用程序是否能根据需要执行操作，即使在承受大量负载的情况下。

1. 确定目的

目的定义了一个人或实体要去满足的条件。你可以给应用程序测试设置各种目的，比如在 42 皮秒内精确计算 pi 的值。当然，这个目的无法达到，因为计算精确的 pi 值是不可能的。有些公司会给应用程序设置相同类型的目的，且在测试过程明显达不到这个目的时放弃。现实的目的是在分配的时间内采用可获取的资源可以完成的。

此外，衡量目的是否达成必须有一个标准。这不只是要知道测试是否成功，还要知道测试成功到什么程度。为了知道测试过程的好坏，你设定的目的必须包含用于定义期望输出范围的方法，比如在给定的 99% 的时间内，应用程序计算出的结果都是正确的。

你给应用程序设定的目的，依赖于你希望从应用程序得到什么以及你有多少时间可用来实现这一目的。但是，你可以依据下面的分类来确定目的。

- 验证和校验
  验证和校验可以发现错误。而且，它还能保证对于给定的特定输入，应用程序可以提供所设计的输出。软件必须按规范定义的那样工作。从安全的角度来说，你必须用期望的和不期望的输入来测试软件，并验证它在这两种情况下是否都能正确响应。

- *优先级覆盖*

  世界上最安全的软件会用每一种可能的方式对每个函数进行测试。但是，在现实世界中，有限的时间和资金使得不可能进行这样完全的测试。为了尽可能完全地测试软件，你必须先进行概要分析以确定软件在哪些地方花了最多的时间，然后将自己的工作聚焦在那里。但是，其他因素也要考虑。即使一个功能不经常使用，但当它是不能失败（一次失败会导致灾难性后果）的情况时，你可能仍然必须给它高的优先级。优先级覆盖必须包含只有你的应用程序会处理的因素。

- *平衡*

  测试过程必须平衡所写的需求、现实世界的限制以及用户的期望。当进行某项测试时，你必须证明结果是可重复的并且是与测试人员无关的。避免测试过程中的偏见很重要。说明规范包含的内容很可能与用户期望的内容不能完全匹配。错误的沟通（或者有时候完全没有沟通）会妨碍在规范与用户认为应用程序应该怎么做之间保持完全一致。因此，你还可能需要考虑将未被写进规范的期望作为测试过程中的一部分。

- *可跟踪的*

  用文档记录完整的测试过程对于后面重新进行测试是很关键的。文档必须描述成功和失败的情况。此外，它还必须指明要测试什么以及测试团队会如何进行测试。文档还要包含测试团队可用来进行软件测试的测试框架和其他工具。如果没有完整记录测试团队用到的每一个东西，就不可能在后面重新创建测试环境。

- *确定性*

  你执行的测试不应该是随机的。任何测试都应该测试特定的软件功能，并且你应该知道这些功能是什么。此外，测试应该给出在特定输入下的特定输出。测试团队应该总是预先了解如何做测试，并且要定义它们应该提供的结果，这样可以使错误很明显被发现。

2. 测试性能

许多人把性能与速度画上等号，但性能不只是包括速度。一个很快但不能正确执行任务的应用程序，是没有用的。类似地，一个应用程序可以很好地执行一项任务，但将其处理的信息提供到了错误的地方也是没有用的。为了良好地运行，应用程序必须可靠、安全、快速地执行任务。

这里的每个性能元素是相互制约的。提升安全性会使得应用程序变慢，因为开发人员要加入更多的代码来进行安全检查。增加代码会使应用程序运行得更慢。类似地，可靠性会导致应用程序变慢，因为更多的检查同样会被添加进来。安全检查会降低应用程序的可靠性，它会牺牲功能来降低风险，一个可靠的应用程序可以在各种给定的环境中提供所有期望的功能（它不会失败）。总之，性能的所有元素之间是相互作用的，如图 11-1 所示。

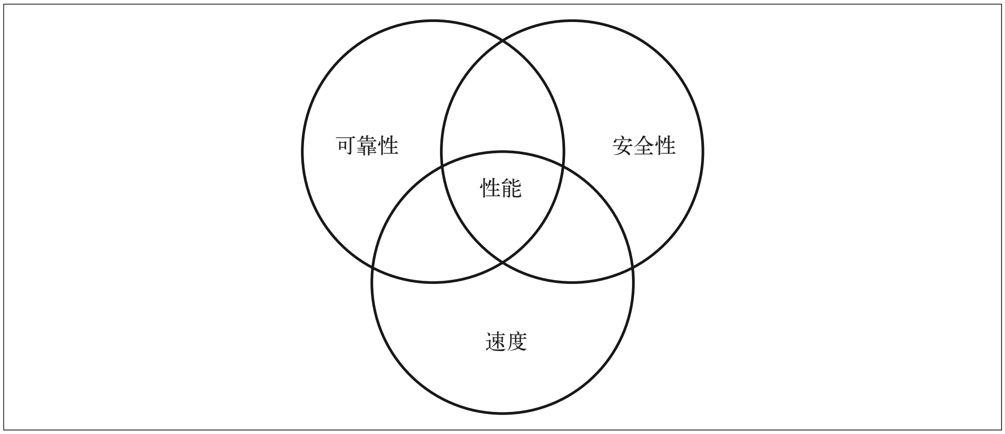


图 11-1：性能包括速度、可靠性和安全性

为了测试应用程序的性能，你必须验证速度、可靠性和安全性之间的平衡。平衡的应用程序可以运行良好且不会给用户增加负担，但仍然可以可靠和有效地处理数据。

3. 测试可用性

许多测试场景没有测试可用性。确定用户能多好地与软件进行交互是很重要的，因为软件的目的是让用户变得更高产（用户是人还是机器不重要）。一个令人迷惑或无用的界面会使用户无法正确地与软件交互，从而会导致安全问题。除了完成任务所需的步骤之外，测试过程还需要考虑用户的身体和感情的需求。例如，让一名色盲用户去点击红色按钮可能得不到想要的效果。没有除颜色以外区分按钮的方法几乎肯定会导致输入问题，而这最终会导致安全问题。

在执行测试步骤时很容易变得自满。用户通常会依赖键盘和鼠标输入作为基本操作，所以你需要测试这两种情况。但是，用户可能有更广泛的访问选项。例如，按住 Control 键的组合会以与单纯按住标准键不同的方式执行任务，所以你还需要测试这种类型的输入。测试每一种可能条件下的每一种输入不是必要的，但你应该知道应用程序可以正确处理各种输入类型。

4. 测试平台类型

所使用的平台不同，软件的表现也会不同。通过本书，你已经看到用户可以并将会使用各种设备来与任何应用程序进行交互。在所有可能的平台上测试应用程序是不太可能的，因为某些平台在测试的时候甚至还不可用。因此，你必须指定平台类型——依据能力和功能，设备会落入特定的分类。例如，根据应用程序的功能，可以将智能手机分为两类或三类。

一家公司不太可能知道用户用来执行与工作相关任务的各种设备类型。在应用程序设计过程中进行一次调查以获取可能的用户设备清单是很重要的。你在为特定的设备类型创建测试场景时可以使用这份清单。

当处理平台类型的问题时，特别重要的是要查看设备在物理上和界面的工作方式有什么不同。浏览器在标准级别上的不同也会造成很大的差异。任何会导致应用程序在不同平台上工作得不一样的问题，都是一个测试主题。你需要确保这些差异不会导致应用程序以你不希望的方式运行。

5. 实现测试原则

测试原则是你可用于各种形式的测试的指导原则。原则影响着应用测试的每个方面，并且每个级别的测试。当你执行 API 的单元测试时，要采用在集成测试时将应用程序作为整体测试所用的相同原则。下面列出的原则对于各种测试来说是通用的。

- 使软件失败

  测试的目标是要导致应用程序失败。如果你测试应用程序是要看它成功，那么永远无法找到应用程序的错误。测试过程应该暴露尽可能多的错误，因为黑客肯定会寻找这些错误并加以利用。你没有找到的错误就是黑客会用来对付你的错误。

- 尽早测试

  你越快找到错误，修复它的成本就越低。修复一个 bug 的成本会随着时间增加，如图 11-2 所示。每个 bug 修复的成本越高，每个 bug 发现得越晚，因为时间和成本的考量，你可以修复的 bug 就越少，应用程序就越不安全。

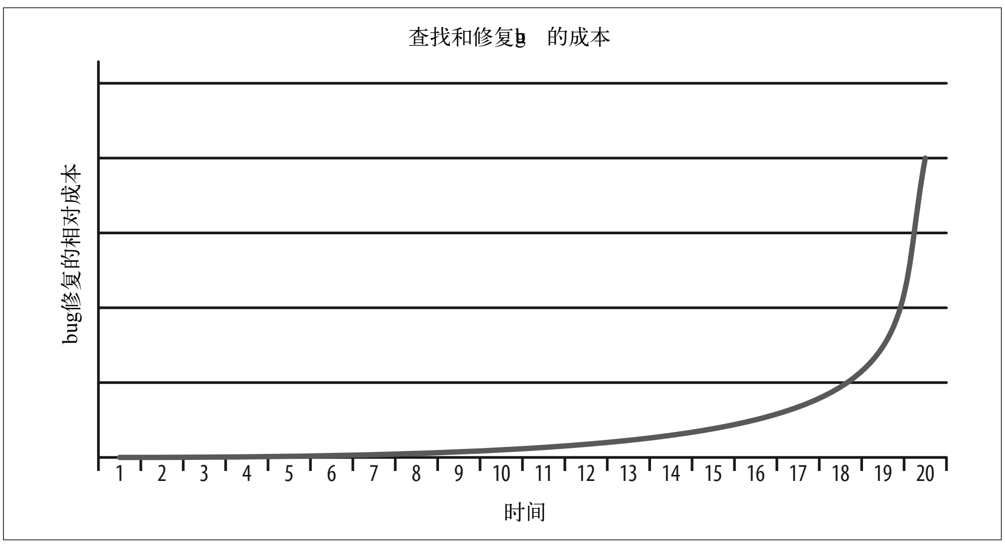


图 11-2：尽早发现 bug 可以留下更多时间和金钱来发现更多的 bug

- 使测试依赖上下文

  应用程序的上下文能帮你确定如何测试它。例如，对安全要求很高的应用程序与电子商务网站的应用程序，需要不同的测试。开发方法也会影响测试上下文。采用瀑布流方法开发的应用程序与依靠敏捷方法开发的应用程序需要不同的测试。使用正确的测试上下文让你更有可能发现导致问题的 bug，从而帮你提升应用程序的安全性。

- 创建有效的测试用例

  创建的测试用例越完整、越精确，测试就会越有效，安全性也会越高。测试用例必须包含用户和应用程序架构的需求。每个测试用例包含提供给应用程序代码的精确输入以及对所期望的精确输出的描述。输入和输出使用可测量的数量以避免歧义，这是很重要的。

- 定期检查测试用例

  重复使用相同的测试用例会创建出无法找出错误的测试框架。随着潜在的应用程序问题变得已知，检查测试用例是很重要的，这样测试才可以继续推动应用程序变得更加坚固并找到更多 bug。作为检查过程的一部分，你还应该执行试探性测试以定位没有人想到过的、没有用户遇到过的以及没有黑客利用过的潜在 bug。

- 使用各种测试人员

  有些公司在发布、验收、集成和单元测试等不同的测试阶段会依赖不同的测试人员。使用分级别的测试是有效的，但是在开发过程中聘用各种测试人员会得到更好的结果。例如，在应用程序的早期阶段依赖用户进行测试能帮忙定位界面设计中的安全问题，在这个时候要修复它们是比较容易和节约成本的。

- 执行静态和动态测试

  使用静态测试可以测出应用程序的深度并反映开发人员对问题领域和数据结构的理解。使用动态测试可以测出应用程度的宽度并反映应用程序处理极端输入的能力。使用这两种测试有助于确保测试过程可以在分配的时间内获得尽可能多的 bug。

- 寻找缺陷群

  错误倾向于成群发生。在一个代码段中找到错误的可能性与在这个代码段中已经发现的错误数量成正比。

- 执行测试评估

  每一个测试用例需要有完成度的评估，以确定其成功或失败。当测试用例的数量低于某个临界值时，你可依赖人工检查员。当测试用例的数量很高时，除了人工检查员，你还必须包括自动化检查。

- 避免没有错误的神话

  不能因为一个应用程序在运行时没有检测到错误就认为其没有错误。应用程序会有各种测试不到的错误，比如不能满足用户的需求。此外，测试只能检查到开发人员为它们创建的检查点。一次测试发现不了某个错误可能只是因为没有测试到它。应用程序通常都有错误，而其中很多是检测不到的。

- 结束测试过程

  理论上来说，你可以持续地测试一个应用程序来查找错误（并继续发现它们）。但是，由于金钱、时间和软件质量等综合原因，测试通常在某个点结束。当使用某个应用程序的风险变得足够低，并且用户认为应用程序足够好用时，测试过程通常会停止，即使仍然有些问题需要考虑。因此，你用于任何目的的任何一款软件都有可能包含导致安全风险的错误。

6. 理解测试的局限性

测试不能提供软件性能的完整情况。它能帮你确定关于软件的具体事实，但不能保证软件不会崩溃。对于测试过程设置实际的期望值并且在理解测试结果并不完整的情况下执行这一过程是很重要的。以下是一些你需要考虑的测试的局限性。

- 测试人员不是通灵大师
  测试只能揭示错误的存在，但永远不能让它们消失。测试过程可以确定已知问题的存在。测试人员无法测试未知或未被发现的问题。
- 测试不是一个决策工具
  测试只能帮你确定软件的状态。它不能帮你确定软件是否可以安全使用或你是否应该带着一些 bug 发布它。
- 用户会找到一个不能工作的环境
  测试只能确定软件会在特定的环境中有效地工作。如果用户在一个条件不同的环境中安装软件，它可能会出故障。事实上，你对环境做的任何改变都可能给软件造成故障，而在测试阶段无法发现这些问题。
- 问题的根源对于测试是不可见的
  测试是要确定在给定特定的输入时故障的影响，它无法告诉你导致故障的根源。在测试过程结束之后你只知道存在一个故障；你必须确定故障原因。

依靠测试工具来弥补不足

人工测试技巧加上测试框架和脚本，让你在定位应用程序潜在的安全漏洞方面有明显的优势。但是，大部分开发人员还要依赖测试工具来帮忙弥补这些长期备用的方案的不足。测试工具可以查找标准的测试可能无法定位的安全问题，比如 SQL 注入、跨站脚本（XSS）、缓冲区溢出以及 flash/flex 应用和 Web 2.0 的漏洞。你可以考虑将以下工具加入到你的开发者工具箱中以弥补不足。

- ♦ WebInspect（http://www8.hp.com/us/en/software-solutions/webinspect-dynamic-analysis-dast/）
  一个动态应用安全测试（Dynamic Application Security Testing，DAST）工具，能自动查找常用的攻击行为。这个工具会模拟在这些攻击下应用程序会如何表现，这样你就能很容易地找出潜在的安全漏洞。这个工具可用于 Web 应用程序和服务。
- ♦ AppScan（http://www.ibm.com/developerworks/downloads/r/appscan/）
  可以执行广泛的测试，包括 DAST、运行时分析以及对于应用程序及服务的静态污点分析。这个工具的重点在于软件生命周期中的弱点管理。
- ♦ Burp Proxy（https://portswigger.net/burp/proxy.html）
  在你的系统上安装一个能拦截应用程序与其他端点通信的代理服务器，使得你可以分析应用程序的请求和响应。使用这个工具可以查找可疑的活动，这些活动平时不会引起注意，但可以作为黑客发起攻击的前兆。

- ♦ Paros（http://sourceforge.net/projects/paros/）
  创建一个代理服务器环境，可以检查和编辑应用程序所有进来和出去的通信。这个工具还能提供一些扫描能力，这可以让自动化测试更容易。

11.1.2 测试内部库

测试内部库（你能控制和拥有的库）时，你可以访问源代码并且可以在组合成库之前对每个部分进行单元测试。这个方法让你对测试过程有更大的控制，可以以较低的成本测试和修复 bug，并确保你可以进行静态和动态测试以更高效地定位潜在的问题。

当然，你拥有代码也意味着你在开发应用程序的同时也在写库的代码。其他的开发人员也会想要测试。这意味着你必须采用 mocking（如第 10 章所描述的那样），以确保开发按预期进行。当你进行了单独的单元测试并确定库的元素按预期工作时，就可以用真实的元素替换模拟的元素。

作为测试过程的一部分，你还可以为库创建测试框架，如 11.2.1 节所描述的那样。但是，不是一次创建一整个测试框架，而是在每次真正的库元素可用时创建一部分。以这种方式创建测试框架有助于你跟踪库元素如何与应用程序交互并根据需要作出改变（当改变的成本很低时）。

11.1.3 测试内部API

与内部库一样，你拥有内部 API 的代码。由于 API 不是应用程序的一部分（它在不同的进程中运行），你需要搭建一套服务器来与 API 交互。但是你不能用生产环境的服务器来执行这个任务，因为你创建的代码会有各种问题。让应用程序可配置是值得的，这样一旦你发布了 API，只更改一个配置项就能让应用程序指向生产环境的 API。

你创建的任何 API 也会需要依赖 mocking，这样应用程序的开发人员可以在你继续完成 API 时开始开发应用程序代码。当真正的代码可用时，你需要用真正的元素替换模拟的元素。意识到模拟的元素会提供罐装回复之后，采用 11.2.2 节中所说的技术，从一开始就为 API 开发一系列测试脚本是值得的。事实上，罐装回复会提醒你哪些元素仍然是模拟的。

很重要的是要用与外部 API 相同的标准来测试内部 API，并且要配置与生产环境匹配的测试和开发环境。否则，你最终可能会使内部 API 成为黑客入侵网络的途径。即使断开连接的软件也很容易遭受各种黑客攻击（如前面章节所述）。

11.1.4 测试外部库

外部库（包括框架和各种类似库的代码结构）是由别人拥有的。这样的库在你开始编写自己的应用程序之前就完成了，并且理论上来说，第三方在测试和维护外部库的代码。但是，这些库在大部分情况下就是一个黑盒。你执行完整的静态测试的能力受限于第三方提供的公开模块。由于处理第三方库的复杂性，完整的静态测试是不可能的，就只剩下动态测试了。

在开始编写自己的应用程序之前，在提案和设计阶段，需要确保你所选择的所有第三方库安全并完全符合应用程序的标准。最流行的库会给你提供一个测试框架，或者你可以找到可用于它们的第三方测试框架。但是，当你使用小型的、不知名的库时，需要自己搭建测试。

> 人们很容易因为一个库、框架、API 或微服务很流行，就认为它是可安全使用的。但即使是像 jQuery 这样知名的产品，你也可以在诸如 http://www.cvedetails.com/vulnerabilitylist/vendor_id-6538/Jquery.html 这样的网站上找到它们的安全问题。此外，即使产品是相当安全的，使用不当也会导致各种安全问题。你需要查看诸如“如何安全并明智地使用 jQuery：一些关键问题”（http://resources.infosecinstitute.com/safely-and-wisely-use-jquery/）这样的文章来发现稍后可能导致故障的问题。每一段代码都会有缺陷，每一段代码都会有使用问题，每一段代码都是不安全的。只要不断重复这三句话，那你就总是正确的。

11.1.5　测试外部API

外部 API 很受欢迎，因为别人拥有代码并且它甚至不用在本地系统上运行。你不需要下载代码或做其他的事情，只需发起调用。调用外部 API 的巨大诱惑会让即使是最小心的开发人员也有虚假的安全感。之前的章节已经告诉你 API 可能会导致灾难。总之，比起其他代码，你需要更小心地测试外部 API，因为不像外部库（扩展到框架），你永远不会看到代码。执行静态测试是不可能的，所以动态测试最好在你决定使用它之前根据其教程运行一遍 API。

与使用库不同，你不太可能找到针对某个 API 的现成的脚本套件。为了验证 API 可以按预期工作，你确实需要创建一套脚本并将各种输入发送给 API。跟踪你接收到的响应是很重要的，特别是对于错误的输入。你不知道 API 会如何响应错误的输入。因此，你不知道要如何编写自己应用程序的代码来对错误的输入反馈作出响应。换言之，你需要知道 API 在接收到超出范围或可能是错误类型的数据时会如何响应。

大部分开发人员认为错误的输入会来自应用程序的用户。但是，API 的错误输入可能来自一次拙劣的中间人攻击或其他形式的攻击。错误的输入还可能来自系统上的其他源，它反映了某种类型的感染或其他问题。意识到 API 会如何响应错误输入，你就可以创建某种安全提示来告诉你某处出错了。把它想成是矿井策略中的金丝雀。错误的输入在大部分情况下不是随便发生的，它是有原因的，而知道导致预料之外的响应的错误输入类型可为我们定位问题的根源提供线索。

11.1.6　扩展测试到微服务

你应该使用与测试 API 一样的技巧来测试微服务。与 API 一样，你只能进行动态测试，除非你拥有微服务的代码。此外，跟踪对预料之外的输入数据的响应是很重要的，特别是当你计划使用多个微服务来执行相同的任务时（替代方案可以对你选择的主微服务提供备份）。你接收到的响应可能在不同的微服务间会有变化，这意味着你的错误处理代码变得更加繁琐。

但是，使用微服务时需要考虑的最大问题是，开发人员故意让微服务保持较小的规模。你不能测试最常用的功能，因为每个功能都是常用的。简言之，测试脚本现在就必须完整测试每个微服务，这会给测试团队增加负担。

11.2 单独测试库和API

第一级别的测试通常是单独使用库和 API。微服务的测试过程与 API 类似，但不需要复杂的测试框架，因为微服务比 API 要简单。接下来的几小节描述了对库和 API（可以扩展到微服务）进行单元测试时可采用的策略。很重要的是要认识到你可以直接测试 API 和微服务，或将其作为 API 虚拟化层的一部分进行测试。

11.2.1 为库创建测试框架

测试框架是应用程序代码中的一组指令，或是专门添加到代码中的，用以执行各种测试。因为库作为应用程序的一部分存在，所以测试库的指令也会作为应用程序的一部分。

测试指令通常出现在调试代码中，比如 `assert()` 这类函数，或者使用日志或屏幕输出。JavaScript 没有 `assert()` 函数（据说要增加一个）。但是你可以使用 `error` 对象来创建一个类似 assert 的函数，来提供同种信息。在使用 `assert()` 时，你要在代码中创建断言，它看起来会像下面这样。

```
assert(typeof myArg === "string");
```

`assert()` 函数看起来如下所示。

```
function assert(condition, message)
{
    if (!condition)
    {
        message = message || "Assertion failed";
        if (typeof Error !== "undefined")
        {
            throw new Error(message);
        }
        else
        {
            throw message;
        }
    }
}
```

在这个例子中，当一个参数的值或其他代码条件失败时，测试会抛出一个错误。你可以选择用日志记录错误或以其他方式处理它，但你要知道测试失败了。测试条件对于任何浏览器来说都不是问题。但是，你可能会发现不同的浏览器会用不同的方式支持 `error` 对象，所以简单地抛出消息（而不是重新抛出 `error` 对象）是一个很好的降级处理。

在代码中的元素会执行测试，但你仍然需要提供输入。为了解决这个问题，你通常需要手动测试脚本，这通常很容易出错，或者用一个第三方脚本来提供所需的输入。你对库进行

的测试会精确地告诉你库能否满足应用程序的需求。

11.2.2 为API创建测试脚本

API 支持发起调用。测试 API 甚至可以不用应用程序。你所需要的就是一段发起调用并检查响应的脚本。很多第三方产品可以执行这个任务，或者你可以创建一个简单的应用程序来手动执行测试。使用脚本可以确保你每次都得到相同的测试结果，所以使用脚本通常是最好的选择。你执行的任何测试最起码应该检查以下条件。

- 范围
  正确响应范围内的值，并且当值太高或太低时给出正确的错误响应。
- 类型
  验证用户提供了正确的输入数据类型，并且当输入的数据类型错误时给出正确的错误响应。
- 大小
  校验数据的长度，这样别人就不能发送一段脚本来替代所期望的字符串。
- 字符
  测试输入中的非法字符以确保用户不能发送控制字符或其他不正确的字符。

当你开始从应用程序使用 API 之后，需要执行集成测试，这包括为应用程序提供输入，然后要求应用程序发起所需的调用。你可以再次使用脚本工具来让任务变得更简单。

11.2.3 将测试策略扩展到微服务

与 API 一样，你在使用微服务时要依靠某种脚本工具来发起调用和检查响应。但是，你需要确保微服务被检查得很彻底，因为每个微服务都代表一段单独的代码。你不能对微服务作出跟 API 一样的假设。

进行集成测试的时候，你需要确定应用程序的性能概况。每个微服务都应该接受至少一次测试。但是，你计划经常使用的微服务应该接受更多的测试。这个策略的目的是要验证有较高概率导致问题的代码，并使测试的成本保持在较低的水平。

11.2.4 开发响应策略

所有测试的焦点在于库、API 或微服务对于给定输入提供的响应。除非代码能做出适当的反应，否则应用程序就不能按原来想象的那样运行。更重要的是，黑客会寻找这种行为上的偏差来加以利用。当定义响应的时候，你必须考虑两种响应类型：直接的和模拟的。接下来会讨论这两种响应类型。

1. 依靠直接结果

直接的响应来自激活的库、API 或微服务。在这种情况下，你会获得针对给定输入的实际响应。如果被测试的代码是正确工作的，你接收到的响应就应该精确匹配定义了测试用例的规范文档。直接的结果可以测试你在应用程序中计划使用的代码。

2. 依靠模拟结果

模拟的结果来自模拟库、API 或微服务的 mocking 软件。使用模拟的结果可以让你在库、API 或微服务就绪之前就可以开发应用程序并进行测试。使用这种方法可以节省时间并能加快开发进度。

但是，这里还有另一种考虑。你还可以使用 mocking 来测试针对库的测试框架或针对 API 的测试脚本的可行性。因为你已经知道对于给定的输入 mocking 软件将会提供精确的响应，所以你能够验证测试软件是否在正确地工作。除非你可以指望测试软件正确地执行任务，否则进行测试不会有什么价值。

11.3 执行集成测试

一旦你测试了应用程序的各个元素，就可以开始将库、API 和微服务集成到应用程序中。最简单、最完整、最不复杂的集成测试方法是采用分阶段的方法，库、API 和微服务以有序的方式小规模地添加进应用程序。分阶段的集成测试可以帮你很快地定位和解决问题。将应用程序从模拟的数据切换到真实数据时，每次只迁移一份可能一开始看起来是在浪费时间，但这个过程可以很快地定位错误，最终会节省大量的时间。集成测试的目的是创建一个能按预期工作的完整应用程序并且不包含任何安全漏洞。

> 你永远不可能创建一个“防弹”的应用程序。每个应用程序都会包含缺陷和潜在的安全漏洞。测试可以消除大部分明显的问题，并且你需要测试（和重新测试）应用程序任何一部分的每个更改点，因为安全漏洞会出现在很微小的地方。但是，永远不要自满地认为自己已经发现了应用程序中所有可能的问题，黑客肯定会给你展示不一样的结果。

开发人员会依赖很多集成测试模型。其中一些模型被设计用于使应用程序尽可能快地运行起来，但只有当没有错误发生时才能完成任务。任何开发软件的人都知道出错在所难免，所以集成测试模型只会导致问题且不能让你完整地检查安全问题。为此，下面列出了三个使用了分阶段方法的测试模型，它们可以比较好地定位安全问题。

- 自下而上

  在这种情况下，开发团队首先添加并测试较低级的功能。这个方法的优点是，对于执行如监控这种任务的应用程序来说，你可以验证你有一个稳定的基础。原始数据在测试阶段的早期就可用，这使得整个测试过程更加真实。

- 自上而下

  使用自上而下的方法会首先测试所有高级功能，接着测试下一级的功能，直到最底层的功能。这个方法的优点是你可以从一开始就验证 UI 的功能正常且应用程序满足了用户的需求。此类测试适用于表现型的应用程序，它们的用户交互性有很高的优先级。

- 三明治

  这是自下而上与自上而下的组合。它适用于那些需要使用一些数据源来执行大部分任务，但用户交互性又是首要考量的应用程序。例如，你可能在 CRM 应用中使用这种方法以确保在实现其他特征之前，UI 可以正确地展示来自数据库的数据。

11.4 测试与语言相关的问题

某些测试套件存在的一个很大的漏洞是缺少对特定语言问题的测试。这些测试可以找出特定的语言在处理特定请求时的缺陷。语言可能可以按所设计的那样准确工作，但是多个因素组合起来可能会产生不正确或预期外的结果。在某些情况下，基于语言缺陷发生的攻击就是以语言设计者原来没有想到的方式使用语言。

每种语言都有缺陷。例如，很多语言都有线程安全的问题。当在非多线程的环境下使用时，这些语言不会有错误。但是当在某些情况下在多线程环境中使用时，这些语言会突然出现问题。它可能会产生不正确的结果或只是以意料之外的方式运行。更隐秘的缺陷是语言表面上在正常工作，但却设法为黑客提供渗透系统所需的信息（比如在线程之间传输数据）。

接下来的几小节描述了最常见的与语言相关的问题，你需要在测试应用程序时予以考虑。在这里，你可以找到 HTML5、CSS3 和 JavaScript 等在 Web 应用程序中最常使用的语言的缺陷。但是如果你在自己的应用程序中使用其他语言，那么还需要检查那些语言的缺陷。

许多测试套件会检查在给定特定输入时的输出是否正确。此外，它们可能会执行范围检查来确保应用程序在其应该接受的值的范围内运作正常，并且当遇到范围之外的值时可以给出恰当的错误反馈。大部分测试套件不检查语言缺陷的原因是，比起你测试的其他部分来说这个问题主要与安全相关。当测试与语言相关的问题时，你真正要找的是缺陷对于应用程序安全性的影响。

11.4.1 设计针对HTML问题的测试

当使用 HTML 时，你需要考虑该语言提供了基础的 UI，同时也为与用户相关的安全问题的发生提供了途径。为此，你需要确保用来展示应用程序所管理的数据的 HTML 经过了测试，以确保它能在各种浏览器上工作。为此，需要考虑以下与语言相关的问题。

- HTML 组织得很好，且不使用大部分浏览器不支持的标签或属性。
- 文档进行了正确的编码。
- 文档里的任何代码要执行必需的错误处理并检查输入是否正确。
- 当提供特定的输入时，文档的输出应该看起来是预期的样子。
- UI 元素的选择要降低令人迷惑并导致错误输入的可能性。

有很多可用于 HTML 测试的工具。其中两个比较好的工具是 Rational Functional Tester（http://www-03.ibm.com/software/products/functional）和 Selenium（http://www.seleniumhq.org/）。这两种工具都能自动进行 HTML 测试并提供测试脚本创建的记录 / 回放方法。Rational Functional Tester 是来自 IBM 的工具，具有专业的测试策略，比如故事墙。如果你需要手动检查一项新技术，可以尝试一下 https://validator.w3.org/dev/tests/ 里的 W3C 标记验证服务。这个网站提供了你可使用的 HTML 版本测试。

HTML 测试的一部分是确保你的所有链接正常工作且你的 HTML 组织得很好。WebLight（http://www.illumit.com/weblight/）等产品可以自动检查你的链接。还有一个类似的工具是 LinkTiger（http://linktiger.com/）。这两个产品都能检查受损的链接，但都提供了你在测试时可能需要用到的额外功能，所以最好先看看这两者的说明文档。

11.4.2　设计针对CSS问题的测试

CSS 的初衷是创造一种格式化内容的方法，它并不涉及表格和其他技巧的使用。这些技巧的问题是没有标准的方法，并且这些技巧会让有特殊需求的人无法使用页面。但是，CSS 已经不仅能用于格式化了。人们发现了很多方法来用 CSS 制作特效。此外，CSS 现在几乎提供了一定级别的编码功能。因此，完整地测试 CSS 变得很重要，就像应用程序的其他部分那样。CSS 有可能会隐藏安全问题。为此，在应用程序的安全测试中，你需要执行专门针对 CSS 的测试——它应该符合以下标准。

- CSS 组织得很好。
- 在代码中没有任何非标准元素。
- 特效不会导致可访问性问题。
- 颜色、字体和其他可见元素的选择要考虑到有特殊需要的人士。
- 处理用户的特殊需要时，可以使用替代的 CSS 格式。
- 给定一个事件或具体的用户输入，CSS 要提供一致且可重复的输出效果。

随着 CSS 使用量的增加，对好的测试工具的需求也在增加。如果你是一名 Node.js 用户，一个比较好的工具是 CSS Lint（http://csslint.net/），你可以使用它来检查代码。当你想要检查外观时，需要能够进行屏幕快照比较的产品，比如 PhantomCSS（https://github.com/Huddle/PhantomCSS）。当网站的屏幕快照以意料之外的方式改变时，PhantomCSS 可帮你辨别改变并查出原因。如果你需要手动校验器来检查想使用的技术，可以使用网页 https://jigsaw.w3.org/cssvalidator/ 上的 W3C CSS 校验服务。

11.4.3　设计针对JavaScript问题的测试

JavaScript 会为应用程序提供大部分功能代码。为此，你要用与测试其他编程语言相同的方法来测试 JavaScript 代码。你需要验证对于一个给定的输入，你是否能得到一个特定的输出。你需要考虑将以下这些问题作为你的测试套件的一部分。

- 确保代码遵循标准。
- 用全面的输入类型来测试代码以确保它能处理错误的输入而不会崩溃。
- 执行异步测试以确保应用程序可以处理在不确定的间隔时间后抵达的响应。
- 创建测试组，这样你就能使用示例的断言代码（或者测试库提供的 `assert()` 函数）来验证大量断言，示例代码可在 11.2.1 节中找到。测试组可以放大使用断言进行测试的效果。

- 验证代码是响应灵敏的。
- 检查应用程序的行为以确保一系列的步骤可以产生期望的结果。
- 模拟失败的条件（比如资源缺失）来确保应用程序可以很好地进行降级处理。
- 执行任何所需的测试以确保代码不会易于遭受最近发生的黑客攻击，这些攻击可能还没有在客户端系统中得到修复。使用针对安全的分析器，比如 VeraCode（http://www.veracode.com/），能够帮你定位并修复一些 bug，这些 bug 可能会让黑客基于最近发现的安全漏洞来入侵系统。

> 你所使用的测试 JavaSript 的工具部分取决于公司进行其他测试所使用的工具和公司使用其他工具的经验。此外，你需要选择一款能与应用程序中使用的其他产品一起工作的工具。有些公司使用 QUnit（https://qunitjs.com/）来测试 JavaScript，因为这个供应商还提供了其他套件（比如 jQuery 和 jQuery UI）。在某些情况下，公司会使用 RhinoUnit（https://code.google.com/archive/p/rhinounit）来进行基于 Ant 的 JavaScript 框架测试。许多专业开发人员喜欢 Jasmine（http://jasmine.github.io/）外加 Jasmine-species（http://rudylattae.github.io/jasmine-species/），因为该测试套件是行为驱动的。如果你大量使用 Node.js，你可能还想要研究一下 Vows.js（http://vowsjs.org/）和 kyuri（https://github.com/nodejitsu/kyuri）。Node.js 开发人员喜欢的另一个工具是 Cucumis（https://github.com/noblesamurai/cucumis），它能提供异步测试功能。

作为规范和设计阶段的一部分，你需要研究的一个问题是现有测试套件的可用性。例如，ECMAScript Language test262 网站（http://test262.ecmascript.org/）可以为你的应用程序提供一些很棒的见解。

考虑攻击的本质

很重要的是要明白很多攻击看起来好像什么也没有实现，因为你并不知道攻击者想要干什么。例如，对传输层安全（Transport Layer Security，TLS）协议的攻击，要依靠 Rivest Cipher 4（RC4）加密算法，这可能看起来更像是某人在尝试使系统瘫痪。如果你是一个尝试弄清楚应用程序正在发生什么的开发人员，就必须知道正在发生的攻击类型，这就是你要确保团队里有一名安全专家的原因。

在这种情况下，攻击依赖于创建带有诸如 cookie 等重复性信息的请求。发送者会在每个请求中包含 cookie，但是每个请求都是加密的，这样信息看起来就完全不一样了。只有在获取足够样例的情况下才可能破解加密并开始从受破坏的浏览器获取数据。

至少有两个团队已经攻破了 RC4，足以用 TLS 从浏览器获取数据。第一种技术大约需要 2000 个小时，而第二种只需要 75 小时。很明显，除非获取的信息有很重要的价值，否则该技术可能并不值得一般的黑客去使用。尽管如此，有这样的技术存在就意味着你需要想想攻击的企图是什么。

这个攻击最有趣的部分是攻击者要依靠 JavaScript 来实现（之前给开发人员留下的印象是只能用来写脚本）。攻击者使用脚本注入或中间人攻击的方式将代码上传到目标浏览器。你可以在网页 http://www.computerworld.com/article/2948937/security/encrypted-web-and-wi-fi-atrisk-as-rc4-attacks-become-more-practical.html 上获取关于这种攻击的更多细节。

要修复这种攻击，得使用更加现代的加密方法作为 TLS 协议的一部分。TLS 支持大量加密方法，包括高级加密标准（Advanced Encryption Standard，AES）。为了保证你的应用程序是非常安全的，你需要确保将服务器配置为不接受 RC4 作为加密方法，而是要求更现代的加密方式。

第 12 章

使用第三方测试

第三方测试涉及使用外部实体来给你的应用程序执行各种测试，包括安全测试。第三方可以提供许多服务，其中有一些是非常全面的。当然，在开始进行任何测试之前，你需要知道你是可以信任相关第三方的。一旦测试开始，需要确保第三方能接收到来自你公司的适当的指令、有正确的监控级别并提供令人满意的输出。第三方可能会执行类似你会执行的测试，而且你要明白第三方的水平高于你自己的公司，否则从一开始就没必要使用第三方测试。

你可能想要依靠，至少是部分依靠第三方测试的理由有很多。其中最常见的理由是时间。然而，许多公司缺乏能力和其他资源来正确地执行一次完整的测试工作。有时候，公司会聘请第三方来保证内部测试人员的诚实，并确保内部测试人员不会遗漏任何东西。使用第三方供应商通常要遵循以下四个步骤。

(1) 找到你想使用的第三方测试服务。
(2) 制订测试计划（将第三方作为一项资源使用），精确定义第三方要如何测试软件。
(3) 在与第三方签约之后实施测试计划。
(4) 基于你从第三方收到的报告和其他输出来验证应用程序。

> 虽然本书花了大量时间处理桌面端、浏览器和手机应用程序，但重要的是，要记住这里所描述的原则也能应用于其他类型的应用程序，它们中的一些几乎总是需要第三方测试。例如，汽车制造厂商本可以通过聘请第三方来确保测试过程完整地测试了网络连接的各个方面，成功避免最近发生的许多联网汽车召回事件（请参阅 http://www.computerworld.com/article/2953832/mobile-security/senators-call-for-investigation-of-potential-safety-security-threats-from-connected-cars.html）。事实就是黑客知道如何入侵这些汽车，第三方测试人员也知道（请参阅 http://www.computerworld.com/article/2954668/telematics/

hacker-shows-he-can-locate-unlock-and-remote-start-gm-vehicles.html）。

第三方测试商还可以检查看起来怪异的攻击类型，这些攻击可能在未来变得非常普遍（请参阅 http://www.computerworld.com/article/2954582/security/researchers-develop-astonishing-webbased-attack-on-a-computers-dram.html）。虽然诸如 rowhammering 漏洞这样的技术在今天听起来很牵强，但可以肯定黑客明天就会利用它们，而一家熟练的第三方测试商可以确保你的软件不易遭受这种攻击。

12.1 找到第三方测试服务

你无法在电话簿里通过“T”字头的记录来找出测试商的列表。如果你能轻易地在黄页里查找到感兴趣的第三方并且知道他们能够很好地测试你的应用程序，那真是太好了。遗憾的是，寻找第三方测试商需要你做大量的搜索，并且要确保你能验证这个第三方能够完成你交付的工作。接下来的几小节会帮你理解与查找第三方测试商相关的问题，并且会描述如何确保第三方测试商会把你的最大利益放在心上。

12.1.1 定义聘请第三方的理由

你可以有很多理由聘请第三方为你进行应用程序的测试。这些理由会由于公司、应用类型和应用程序解决的问题领域的不同而不同。在你聘请某个第三方进行测试之前，很重要的是知道你聘请他们的原因。如果你没有这个关键的信息，那么测试工作从一开始就注定要失败，因为你对测试过程有着未言明的和可能不切实际的期望。总之，因为你一开始就不知道自己的期望，所以也无法知道测试是否会成功。为此，最常见的聘请第三方的理由包括以下这些。

- *质量*
  雇用专业的测试人员可以提高软件质量，因为专业的测试公司会知道你的问题领域中所有最新的测试技术。这些测试人员每天都会执行测试，所以他们知道软件在使用某一具体的编程语言、操作系统、平台和解决特定用户需求时通常会出现的 bug 和问题。你在测试过程中接收的报告很可能也要好于自己公司输出的任何报告，因为这些测试人员每天都要填写这种类型的报告。

- *成本*
  比起兼职的人员来说，专业的测试商可以更快且更加准确地执行测试任务。这些测试人员不会对你的应用程序进行无用的测试，这个问题在公司里很常见（既浪费时间又浪费金钱）。此外，也可以将测试外包到世界上的某个地方，在那里聘请他人做测试的开支要远远小于你所在的地方。

- *培训*
  许多公司没有考虑到的一项隐性成本是培训团队执行测试任务所需的时间和开支。就算团队成员是兼职执行该任务，他们也没有专业测试者的经验，并且在面对每种新的应用程序时可能需要额外的培训。唯一值得拥有真正的测试团队的情况是，你的公司编写了

足够多的软件来让这个团队一直从事测试工作。

- 时间
  搭建测试实验室，培训测试人员并等待他们成为熟练的测试人员，这需要花费对于许多公司来说不能接受的时间。为了使产品准时上市，需要在编码阶段就使用尽可能高效的技术，在尽可能接近生产环境的系统中进行测试，且将相关报告反馈给开发人员以便尽快定位和修复问题。

- 专业技能
  你的公司可能最近对平台、设备或其他一些资源进行了更新，而这些更新需要新一轮的测试。应用程序或相关的服务没有更改，但它们运行的环境有变化。如果你发现更新给你当前的测试团队带来了问题，那么可能需要第三方来帮助发现这些改变对于软件来说意味着什么并建立新的检查机制。第三方还能提供许多所需的培训。

12.1.2　考虑测试服务的范围

聘用第三方测试商的核心优势是能够确保从所提供的服务中获得充分的价值。但是，世界上最好的测试服务也无法知道每一种可能的测试的所有信息。你需要精确定义想要完成哪种类型的测试，然后依此选择测试公司。第一步，你需要定义环境，包括以下几项。

- 平台
- 操作系统
- 编程语言
- 问题领域
- 公司类型
- 用户类型

不是每一个测试公司都能执行每一种测试。一旦确定了环境因素，你就需要考虑所需的测试类型。为了获得最佳的结果，很重要的是第三方测试商要提供执行你需要的所有测试的设施。以下是一些最普遍的测试类型。

- 安全性测试
- 功能测试
- 自动化测试
- 性能测试
- 集成测试

> 现在有很多各种级别的测试类型。你可以在网页 http://www.softwaretestinghelp.com/types-of-software-testing/、http://www.aptest.com/testtypes.html 和 http://www.testingexcellence.com/types-of-software-testingcomplete-list/ 上找到更长的测试类型清单。但是，这些清单也不是完整的。记住，当你拥有软件时你还需要考虑静态和动态测试技术，而这两种测试都可以请第三方测试商来做。

有些测试公司只提供一个服务，那就是测试。但是，为了使开发任务成功，你可能需要其

他服务。一个提供全面服务的公司通常比只提供测试的公司收取多得多的费用。服务的数量不一定会影响盈亏总额，但有可能会影响，所以你需要只选择那些你真正需要的服务。以下列出了一些常见的附加服务。

- *开发人员辅导*
  在某些情况下，测试可能会出现开发人员根本没遇到过的问题。开发人员辅导可以帮团队更快地定位和解决问题。此外，它还提供了专门针对应用程序的培训，这意味着开发人员所学到的信息是仅与公司相关的（这提升了它的价值）。

- *程序管理*
  你可能需要获得一些帮助来管理测试工作。正如开发团队可能缺乏测试经验，管理团队可能缺乏管理产品生命周期某些方面的经验。有些测试公司可以帮你的公司解决这个问题。

- *在线学习*
  整个公司可能需要一些培训来让应用程序工作得更好。例如，许多用户不了解安全方面的问题，所以他们会犯导致安全漏洞的错误。在线学习模块的使用可以帮用户在想要并且可以完成培训的时候，学到成功使用应用程序所需的技能。

考虑对应用程序进行安全测试时的细节也是很重要的。第三方测试商可能会提供某些类型的安全测试，但其他类型的则不会提供。此外，你需要知道第三方应该提供什么。以下是一些值得考虑的事项。

- *环境*
  你需要定义测试是针对预备环境的还是针对生产环境的。预备环境测试最适合应用程序开发的早期阶段。生产环境测试最适合最终的开发阶段和应用程序被首次发布到生产环境时。

- *测试级别*
  有些测试商会进行基本的测试以确保应用程序的安全风险较低。但是，这个基本测试可能不会提供足够的信息来修复应用程序包含的问题。你可能希望测试能用示例代码提供对漏洞的充分证明，这样你就能精准地发现黑客是如何工作的，从而可以进行更好的修复。（第三方有时候也会帮忙对你的应用程序进行一些问题鉴别。）

- *资源的使用*
  测试要使用系统资源。为了防止在测试期间出现应用程序功能或可用性的损失，你需要指明资源使用规则。第三方测试商可以配置测试工具来遵循你设置的指导原则。此外，请确保结合时间指定资源的使用规则。例如，你可能想让第三方测试商在晚上或周末进行测试，此时测试系统的使用量较小，而你可以放宽某些资源使用规则。

- *测试停止的条件*
  你肯定不希望安全测试导致数据泄露或其他安全问题。在某些情况下，安全测试确实可能会导致安全问题。定义停止条件有助于防止测试场景变成你需要进行数据修复或公开声明发生数据泄露的条件。

12.1.3 确保第三方是合法的

任何人都可以对你说他们的公司可以及时提供优越的测试服务，并且费用最低。这并不意味着他们真的能兑现承诺。这可能只意味着他们有一名优秀的市场人员，可以写出高于一般水平的宣传材料，但是没有一项会真正得到实现。一个糟糕的第三方测试商会让你损失惨重，它可以导致你的竞争对手率先交付产品，导致数据泄露，从而使客户流失或使得公司员工无法工作。因此，确保供应商真正能兑现承诺是寻找供应商的关键。

安全测试是你真的不想冒险的几个领域之一，所以要确保你聘请的第三方有所有必需的认证。美国软件测试资格委员会（American Software Testing Qualifications Board，ASTQB，http://www.astqb.org/）是查找第三方所需认证的一个好地方。另一个可以查找的地方是国际软件认证委员会（International Software Certifications Board，http://www.softwarecertifications.org/）。无论你选择哪个组织，都要确保你聘请的公司拥有具备适当认证的测试人员。此外，你要得到书面的保证，只有经过认证的测试人员会在你的项目中工作（否则该公司可能会用不符合标准的人员顶替，这些人只是受经过认证的人员的监督）。

考察供应商的稳定性也是值得的。年度报告会告诉你许多关于公司健康状况的信息，也会告诉你公司是否能成功地完成任务。成功的公司会为你的应用程序带来更好的测试，而你希望获得最好的结果。你还应该深挖这些公司的统计信息（如人员流动，这表明了职员的开心程度），并要确定公司是否有过负面报道。

查看这些公司之前顾客的评论也是有帮助的。但是，重要的是要记住评论只是意见且客户有时候是带着非常大的偏见的。此外，一家公司可能会找人发布言过其实的正面评价。可能的情况下，请检查评论并确认客户真的请这家公司提供过服务。联系相关客户，了解该公司完成了什么类型的测试（当然，是在可能的情况下）。

12.1.4 面试第三方

此时，你已经知道自己为什么需要第三方的帮助，你也已经找到了能提供所需服务的公司，并且该公司是符合标准的。接下来就要和该公司的人讨论一下，确定你是否能与他们建立合作关系。这并不像是一场为自己公司招聘职员的面试，事实上，你是在聘请整个公司来为你的应用程序进行测试以确保它是安全的。这意味着要让公司里的所有利益相关者参与其中并确定这项工作的最佳候选人。这些利益相关者当然包括管理层，但是也包括开发人员、培训人员、用户和任何受应用程序影响的人。你需要回答的问题是你正在聘请的公司是否真的能完成所要求级别的测试以获得一个可用的应用程序。如果你需要其他服务，那么你还需要问问该公司能否提供你期望的支持（如果该公司提供培训支持，用户就能马上变得熟练）。

12.1.5 对测试的搭建进行测试

在合同上签字之前，要确保你知道你聘用的公司可以完成工作。如果可能，让该公司提供某种有限的示范，证明他们可以完成一项普通的测试流程。你应该能够看到一些演练或一些其他的证据，表明该公司确实能完成工作。否则，很难知道你得到的会是什么，直到事

情变得无可挽回。让这些公司展示一下其熟练度通常是一个好主意，即使只是对它们计划采用的测试过程的预演。

12.2 创建测试计划

如果没有测试计划，是不可能测试任何软件的。你在之前的章节中也看到了这个要求。但是，当聘请第三方时，创建一个普通的计划不会有太大效果。你的测试计划需要说明你期望第三方测试商做什么。这个文档需要把你公司的工作和测试商的工作区分开。此外，这个计划必须考虑在这些条件下，什么是成功的测试。你不必担心应用程序的失败状态，而需要考虑公司与第三方测试商之间的合作是否能满足特定的目标，以确保应用程序是相当安全的。接下来会描述如何创建一个测试计划。

12.2.1 指明第三方在测试中的目的

第 9 章讨论了在应用程序测试中定义目的的需要。第 11 章进一步强调了定义目的的必要性并具体描述了目的。但是，这些是对应用程序作为一个整体和公司作为一个整体的目的。第三方测试商是在测试过程中的合作伙伴，这意味着其只参与测试过程中的一部分。第三方测试商的目的与你类似，测试仍然必须反映特定的原则；但是，目的现在还包括清晰地交流所有的测试标准和方法。此外，你必须提供包含公司管理层策略相关概念的目标。因此，第 9 章和第 10 章描述的目的是一个好的开端，但现在你必须增强它们，以确保第三方确切地明白他们需要做什么且知道如何响应报告信息的要求。

12.2.2 创建书面的测试计划

你通常需要创建一个书面的测试计划。但是，当聘用第三方测试商时，你必须仔细检查这个书面测试计划以确保第三方测试的作用是清晰的，并且第三方完全理解了所有的测试需求。如果有存在问题的地方，第三方测试商应该可以提供反馈来帮忙澄清测试需求。事实上，这个反馈作为对未来测试计划的培训是非常宝贵的。第三方测试商应该可以帮你制订一个完全遵循行业最佳实践的测试计划，并帮你从测试过程中获得精准的结果。

作为测试计划的一部分，你需要用第三方测试商的服务清单作为输入来列出要进行的测试。换言之，测试计划应该清楚地声明测试过程中包含第三方测试商的哪些服务，这样一旦测试开始就不会有模糊地带存在。确保你精确地理解要执行哪些测试以及为什么它们对于确保应用程序按预期工作是必需的。第三方测试商必需能够清楚地表达测试和其他服务的需求。否则，测试过程可能会由于缺乏沟通而失败。

> 很重要的是，要在初始阶段与第三方测试商建立信任关系。当然，信任来自于了解第三方测试商真的明白如何在你的问题领域进行应用程序测试。验证第三方测试商是否符合标准是一个好的开端，但是请确保专家已经就位以监督任务，这些任务指明测试过程应该如进行。仅当你知道第三方测试商真的理解你（并且不会企图以某种方式欺骗你）时，从第三方测试商那里获得的输入才是有用的。

12.2.3 枚举测试输出和报告的要求

测试计划中必须精确地描述期望输出。此外，测试计划必须详细描述当测试失败时要做什么，这样响应就会是有条理的，且在测试过程开始之前就已考虑好。第三方测试商通常会帮忙提供关于期望输出的额外信息，并且可能会提供额外的测试类型，这样你能验证应用程序对于最新的威胁是否是安全的。你的员工不太可能知道所有潜在威胁，所以第三方测试商的一部分工作是确保你除了已知道的安全问题之外，还能够测试你所不知道的安全问题。测试计划应该包含对于每一项测试的详细输出。

当测试周期完成或某些其他事件发生时，测试计划应该指明某些类型的书面报告。这个报告不需要呈现在纸上，只需要是一种持久性的形式，让每一个人都能够根据需要检查和学习。这些报告通常与测试或其他重要的事件关联，而测试计划应该详细精确地描述哪些测试要输出哪些具体报告。

> 确保你能从第三方测试商那里得到报告样例的副本。确保你能理解这些报告的内容。在许多情况下，第三方测试商可以改变报告来更好地应对你所在公司的测试类型。此外，你可能需要特定格式的报告来满足监管或法律的要求，所以要确保第三方测试商了解这些需求并能提供所需格式的报告。

12.2.4 考虑测试需求

测试不仅仅是在特定环境下进行的简单输入和输出。作为测试计划的一部分，你必须考虑测试公司如何进行测试。在某些情况下这会成为一个关键点。例如，你可能只有美国一个市场。如果测试公司是在香港，他们习惯于讲英式英语。虽然区别很小，但在某些情况下足以使测试变得无用。考虑解析那些在两种英语格式中有轻微变化的词语，比如 color 和 colour。你的代码可能不会正确地解析 colour。反之亦然。测试公司需要支持所有你想要支持的语言。

从各种角度考虑你要如何测试一个应用程序是很重要的。例如，如果应用程序是设计用来满足特定的健康需求的，而测试公司不够重视可访问性测试，那么应用程序可能会提供所有的东西，但最关键的可访问性却没有。如果用户不能与应用程序交互，那么其是否安全就没有意义了。

制订测试计划的这一部分是特别困难的，因为你已经培训过公司的所有测试人员去寻找应用程序的特殊需求。你很容易会假设测试公司会基于你已经做的事情去看待测试。关键在于在测试时不要做任何假设，而要把所有东西都写下来。

12.3 实施测试计划

一旦你有了一份测试计划，就需要考虑如何实施它。拥有目标是一个好的开始，但是提供一个过程来达到这些目标是成功完成测试所必需的。当实施测试计划的时候，你必须考虑执行监控的需要。这不是一种窥探，而是一种指导——为了帮助第三方按你期望的方式进行测试，你必须提供必需的方向性指导。实施计划的行为也会导致意想不到的问题。你需

要一些处理这些问题的程序，否则测试可能会慢到停下来。接下来描述实施一个测试计划所需要的技术。

12.3.1 确定公司参与测试的程度

让公司参与到测试过程中来是很重要的。否则，你不能确保测试公司在正确地执行任务。但是，监控测试过程与实际参与有很大的差别。你甚至可能会把测试过程分解为几个功能区域：公司可以执行测试的功能区域和测试公司擅长的功能区域。测试计划应该明确地说明每个实体的参与程度。

实施测试计划通常表明写在纸上的最佳计划仍然缺乏实际的用处。你可能会发现测试计划没有包含实际参与的每个领域（事后要让这些领域分解），或者没有考虑到以特定方式进行测试的逻辑。测试计划可能没有充分定义所需的交流级别，或者你可能发现交流过程是闲谈式的，浪费时间。进行一次测试过程的预演通常能告诉你相关的问题。

12.3.2 开始测试过程

在计划和预演之后，就要开始测试过程了。你可能以为第一天会很顺利，但你会失望的。明智的做法是假设在测试过程中会出现一些问题而你需要率先处理它们。这对于开发者来说意味着要尽可能快地与测试公司沟通源代码文件缺失等问题。你还可能发现测试公司无法访问运行应用程序所需的在线资源，比如库、API 和微服务。最佳的做法是把测试的头几天空出来用于解决系统问题，这样，紧急启动测试就不会使你面临的问题更加复杂。

根据应用程序配置、目标和资源，你还应该计划解决一些安全问题。测试公司可能无法访问付费的在线资源，或者你可能要为公司拥有的资源提供额外的权限。最低限度是要确保让所有需要的人员在测试的前几天都到位，使应用程序可以正常工作。例如，如果应用程序需要访问数据库，请确保有一位 DBA 可以帮忙提供所需的访问级别。协调工作最好交给开发运营，如果公司有负责这个任务的人员。不要忘记包含必要的管理层人员。

> 永远不要让测试公司随意访问公司资源。测试公司应该只获得可以使应用程序运行所需的最小资源。此外，你应该为这些资源提供最低的权限。如果测试人员没有要求管理员级别的访问权限，那么你就应该将访问限制在合适的用户级别上。限制访问的主要理由是保护你的资源。但是，就算测试公司是完全值得信赖的，让测试人员在检查应用程序时使用与其他每个人相同的权限也是很重要的（在安全测试时一个特别重要的要素）。

12.3.3 执行必需的测试监控

监控测试公司是确保测试按计划进行的关键。你需要确保测试公司执行了所有需要的测试（而不是简单地签字），并且没有执行任何未授权（并且可能需要收费）的测试。在测试期间监控资源也是值得的。例如，DBA 应该确保测试过程不会破坏任何数据库数据，即使数据是专门用于测试目的的。有些应用程序的故障不会立刻显现，除非你执行了必需的监控。

12.3.4 处理意外的测试问题

测试是一个对抗的过程。毕竟，你正在试图迫使应用程序失败。任何对抗的过程都会在某些时候产生意外的结果。你可以花尽可能多的时间来陈诉一个给定的输入应该产生一个特定的输出，但是这句话的重点在“应该”这个词。你要进行测试的理由就在于要找出结果与你的预期不一样的时刻。

当然，测试人员会使用你拥有的任何报告 bug 的系统来处理大部分意外结果。但是有一些意外的结果来自外部因素，比如团队人员之间的错误沟通。在这种情况下，将问题当作 bug 报告并不会有什么帮助，因为问题不是来自应用程序内部，而是外部因素作用于应用程序或测试过程的结果。为了处理这种测试问题，你需要一个 bug 报告系统之外的工作流程。

处理这种问题的一种常见方式是举行一次临时会议。问题在于会议会变得不聚焦，难以排期，并且可能不会产生任何有用的结果，因为利益相关者需要时间去研究问题。有些公司依赖内部论坛或 wiki 来处理这类问题。事实上，有些产品可以不同方式来解决这个问题。你可以在网页 https://blog.profitbricks.com/top-25-collaboration-tools-for-developers/ 上看到 25 个常用的开发者协作工具。

12.4 使用结果报告

第三方测试商可能简单地告诉你你的软件相当了不起，并且可以完整无瑕地工作，这在开发者社区是很少见的。但是，口头报告不会为你提供后面进行分析的方法，或者创建计划以修复第三方测试商发现的问题的方法。你需要各种报告来向公司的不同成员展示结果。这其中的许多成员没有编程技能，所以你需要一些报告来简化事情，还需要一些报告来提供详细的分析以便进行修复操作。接下来的几小节描述了如何利用报告，以有效方式使用第三方测试结果。

12.4.1 与第三方讨论报告输出

在测试阶段描述应用程序性能的报告会发生变化，而涉及的各方也会发生变化。例如，现在大部分应用程序依赖第三方库、API 和微服务。你可以选择将这些第三方元素的问题报告给开发它们的供应商，作为使应用程序工作得更好的一个方法。当然，你不希望将完整的报告（包括关于你个人代码的报告）分享给这些第三方供应商。第三方测试商可以与其他第三方联系，但你需要有某种策略来管理这个问题。如果你让第三方测试商来执行这个工作，那么第三方利益相关者会简单地上传他们最新版本的代码给测试人员，这极大地简化了你的任务。（你仍然需要知道对第三方代码做出的任何更改，因为这些改变会影响到其他应用程序。）

测试人员创建的报告需要针对公司里不同的读者。管理团队可能只想看到提供状态信息和好看数据的概述。用户可能只想知道影响可用性的问题以及应用程序的界面类型。开发人员想要详细的报告，其中指明准确的错误位置和可能的修复方法。DBA 可能只想知道影响数据库的具体问题。总之，报告要有不同的形式来满足特定读者的需求。

12.4.2　向公司展示报告

应用程序的测试过程会自动产生绝大多数你所需要的报告。路由机制可以确保每个需要报告的人可以通过某种方法得到报告，比如电子邮件。但是，把报告交给利益相关者和让他们阅读报告并不是一回事。当然，问题是为什么每个人都应该关心这个问题，毕竟利益相关者才应该阅读这些材料。问题在于利益相关者会忽略这些信息，直到他们真的需要的时候，而此时对于应用开发来说通常太迟了。为了让应用程序在测试结束时能按预期运行，利益相关者需要参与其中。

根据所使用的开发模型，你可能需要每周向公司展示报告。有些模型需要在周一召开会议。每个周一你要展示一份包含项目进度、任何问题以及一个示范的报告。能够每周看到进度的利益相关者会更乐于参与到项目中并提供更好的输入，这意味着在项目后期（无法添加功能的时候）被要求添加功能的问题会减少。

将周一作为展示报告的日子还能给开发、测试和其他团队时间去响应对于应用开发的建议或需求。不同的团队有一周的时间来开发和更新应用程序版本，让利益相关者在周一大吃一惊，而不必在周日进行紧急修补。因此，各个团队都能获得一些休息时间并做出更好的工作。



最不宜向公司展示报告的日子是周五。首先，没有人会关注任何东西，因为他们已经开始考虑怎么过周末了。就算他们关注，在经过一周的工作之后他们都很疲惫并且不太可能提供有创造性的反馈，而这些反馈是你设计能有效工作的应用程序所需要的。当刚知晓修复需求时，不同的团队还会感到有义务周末加班来修复缺陷。如果你想得到对应用程序有用的反馈，请避免在周五作报告。

12.4.3　根据测试建议采取行动

测试、报告、讨论等的结果是各个团队需要修复不同的测试人员发现的缺陷。有些问题比其他问题更加重要，并且最有可能在应用程序布置之后导致问题，所以你要首先处理这些问题。测试团队和利益相关者交互的一部分就是给每个问题定义其重要程度。例如，在修复导致应用程序几乎在每个人的机器上都崩溃的缺陷之前，你不会想修复一个低优先级的问题。

但是，只是修复问题还不够。在某些情况下，一个问题有不止一个答案，而你可能要创建实现每个答案的应用程序版本来确定哪一个是最好的。不同的利益相关者需要检查潜在的修复来决定哪一个最好（可能不会是相同的修复方案）。为一个问题创建并执行可能有用的修复方案是根据测试建议采取行动的一部分。

把修复交给测试人员测试也是很重要的。但是你可能要决定不立刻测试修复。可能一个交互会使得在没有修复应用程序的其他问题之前难以或不可能测试当前的修复。很重要的是要查找问题的依赖项并确保这些依赖项与原问题一样得到解决。

第四部分

实现维护周期

软件可以并且确实会受到破坏。即使没有任何事情发生，软件也会变旧或停止与新硬件交互。在某些情况下，用户会抛弃软件或发现软件不再满足他们的需求。你今天提供的新的、令人兴奋的、闪闪发亮的代码会很快变成过时的、衰败的昨日代码，如果它足够有名，就会成为让人惊叹的“博物馆珍品”。在你花了大量的时间和精力之后，听说你的代码最终被扔到垃圾堆里是很难受的。当然，通过维护代码，你可以完全避免这种情况，这也是本书这一部分的重点。如果你确实想让你的应用程序尽可能长时间地提供服务，你必须从一开始就有保持它活力的计划。

本书的这一部分会将维护软件的任务分解为三个不同的部分：确定何时更新软件（第 13 章），确定如何进行更新（第 14 章），以及使用报告来监控更新要求（第 15 章）。通过遵循这个三部分流程，你可以让应用程序保持活力，从而成功地让人们继续喜欢并使用它。

第 13 章

明确定义升级周期

应用程序会不断地暴露在网络威胁之下，它所依赖的每一份代码也是如此。为了减少已知威胁带来的风险，需要定期对应用程序进行升级。以下这些升级措施会从几个方面改善应用程序，它们全都对安全性有影响。

- 提升可用性以减少用户出错的情况
- 修复编码错误
- 提升速度以防止用户做出意料之外的事情
- 修复功能性代码以减少潜在的漏洞
- 进行必要的修复以升级第三方库、API 和微服务

本章从多个层级来看待升级过程。你不能只是被动地进行修复，而应预见来自各方的问题，特别是由愤怒的用户引起的问题。升级需要一定量的计划、测试和实施。以下几节将讨论升级的这三个层级，并会帮你制订对应用程序有意义的计划。

13.1　制订详细的升级周期计划

为实施变更制订计划是应用程序升级流程的一部分。问题在于，你通常不知道第三方软件所需要的升级，不理解进行升级的要求，无法按具体的顺序进行升级，然后在实施升级计划时遭遇各种问题。关键是要创建一个有序的方法来评估升级，这就是为什么需要制订升级周期计划。图 13-1 显示执行这个任务时可以遵循的流程。

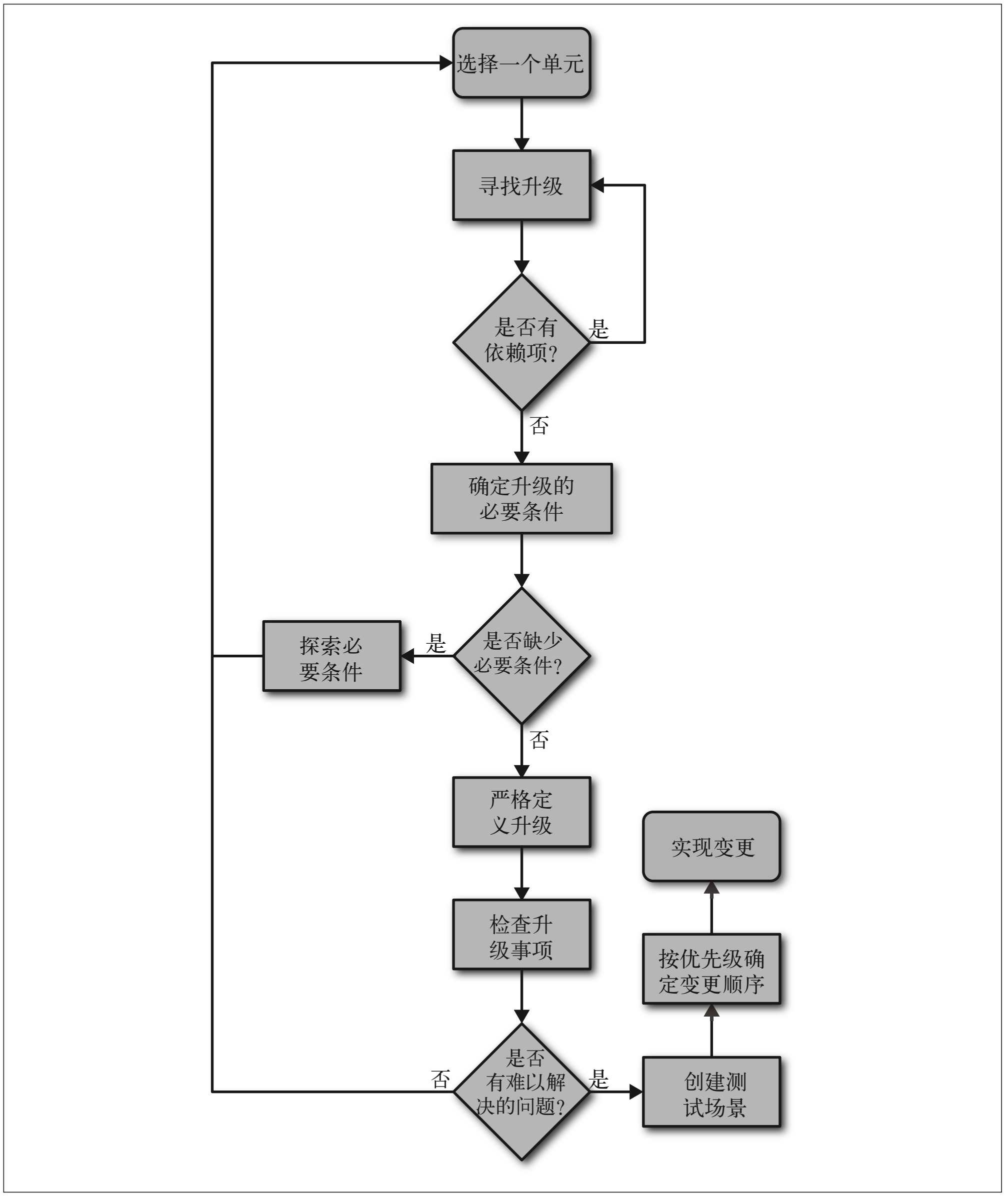


图 13-1：在制订升级周期计划时遵循有序的流程可以节省时间和金钱

这里的要点是每次只处理一个单元且确保你测试了与之相关的一切内容。即使你在其他单元测试中测试了某个依赖项，也必须在当前的单元中测试它，因为当前的单元可能以不同的方式使用该依赖项或以不同的方式影响它。

了解可以遵循的流程之后，来详细看看如何执行这个任务。接下来会讨论与升级周期计划有关的问题。读完本节之后，你应该能够更好地理解如何制订满足你的应用程序需求的计划。

13.1.1 寻找升级

由于用户投诉和系统中断而由支持人员收集的故障单通常会告诉你需要对你应用程序代码做什么样的升级。如果故障单没能告诉你所需的所有信息，它们至少为你提供了足够的信息去进行必要的搜索，以准确发现你所需要的升级。但是，第三方应用软件有不同的情况。除非开发者或利益相关方在调试内部代码时定位了问题，否则升级是不会发生的。查找第三方软件的相关升级并确保你有它们的完整清单是很重要的。否则你无法维持应用程序的安全状态，造成安全漏洞的未知风险也会增加。许多公司最终会因为没有意识到要进行必需的升级而遭受打击。

> 有些第三方提供了自动更新通知。自动更新通知是一种确保你知道它们的方式，但你不能依赖它们。通知会告诉你绝对可用的更新，但你还必须去积极检查你没有听说过的更新。即使一个供应商致力于提供通知，这些通知仍然可能会丢失或被放入垃圾文件夹。

当使用公开库、API 和微服务的时候，检查供应商的网站可以了解到你需要集成到应用程序中的升级。许多情况下，供应商会提供 beta 版本的升级，这样你就能比较早地使用它们。花时间检查升级会造成什么影响是值得的，这样你就能知道这个升级对你的应用程序的重要性。与所有影响应用程序所使用功能的升级相比，对应用程序使用的功能没有任何影响的升级会显得没那么重要。但是你仍然应该在某个时间点进行升级，即使确定对你的应用程序没有什么影响，因为升级常常有依赖性的交互，即使是供应商自身也可能不知道（或者是没有完全测试）。

行业刊物中的文章和供应商发布的文章也会告诉你即将进行的升级。此外，你可以留意那些讨论你所使用的产品的缺点的文章以及供应商修复它们的计划。所有这些信息都要反馈到你的升级计划中。提前了解会发生什么，你就可以提前考虑升级问题。

当要进行升级时，不要忽视人员沟通方面的影响。通过供应商内部的人员，你常常可以比其他人早知道升级的信息。提早知道升级会给你带来竞争优势，因为你可以比其他人更快速地准备升级。在计算机行业，时间是一个很重要的资源，那些一旦需要就能发布升级的公司会是能留住客户和保持吸引力的公司。能够制订计划也会减少员工的压力，并可以让每一个人进行更好的升级（贸然的升级有时候会带来比它们所修复的问题更大的麻烦）。

当听到需要进行升级（无论是对自己的代码还是对第三方提供的代码）时，你需要把相关信息列在一个清单里。开发团队的每个人都应该能访问这份清单并了解未来升级的要求。鼓励你的团队去讨论潜在的升级以及这些升级会给应用程序带来的影响。当时间因素很重要的时候，让人们思考升级以及如何进行升级，都会减少你后面的工作量。

13.1.2 确定升级的要求

知道一次升级的细节是一个好的开端。但是对于做任何具体的升级来说，这还不够。你必须确定进行升级的要求。在某些情况下，一次升级可以带来与创建一个新应用程序同样的问题。预测一次复杂的升级会花费多少时间和需要多少资源，几乎是不可能的。制定升级要求规范时，你可以考虑以下内容。

- *确定资源的要求*

  不是所有的升级都需要额外的资源，但令人惊讶的事实是，大部分都需要。资源可以包括各种东西：内存、处理周期、硬盘空间、数据源、屏幕空间、输入类型（比如传感器）、网络带宽等。这份清单可以变得很长，而你需要仔细考虑它，因为资源的改变可能有副作用，比如在用户过去依赖的设备上无法再运行应用程序。

- *获取任何所需的平台变化*

  留意被废弃的特性始终是一个好的做法。被废弃的特性会导致各种问题，因为有时候它们不会考虑比较旧的平台。你需要仔细考虑一个平台的变化会如何影响用户。可能一次升级会将比较旧的、但仍然热门的、人们喜欢的操作系统排除在外。不要去听无休止的对平台改变的抱怨，你需要在实施它之前了解它以及它对你的公司、应用程序和用户的影响。

- *考虑创建、测试以及发布升级所需的时间*

  需要考虑的升级要求中最难的一部分就是创建、测试和发布升级所需的时间。无论你多么仔细地尝试计算出这个要求，它通常都不会符合实际情况。大部分人都预测他们能用比真实所需时间短得多的时间完成任务。当确定这个升级要求时，请确保用真实数据，并为那些在软件项目中经常发生的意外事件增加时间和资源。

> 当尝试计算时间要求时有很多问题。你很少会在书中看到的一个问题是，需要考虑人员因素。你团队中的人员会被很多项目中的任务牵拌着。如果他们觉得可以把你的项目搁置一段时间而去完成更重要的任务，他们肯定就会这么做。这就是拖延因素，而在计划过程中你需要用一些积极的方式来处理它。除非团队成员认识到你的项目的重要程度，否则每一步的拖延因素都会给你制造问题。就因为在计划中没有关注人员因素，你本来能够按时完成的项目会突然变得需要延期。

- *创建一份进行升级所需的人员和技能的清单*

  升级项目的一个普遍问题是，原来的计划没有包括完成这项任务所需的人员和技能的清单。人们通常认为一小部分人或可能只需要一个人就能成功完成升级。遗憾的是，并不是如此。升级要像新项目那样仔细规划每一点。为了确保升级达到相同的高标准，你需要拜访原来项目里的那些专家。

> 升级时常常会忘记将用户作为团队的一部分。因此，升级会破坏 UI，并导致用户很快就会发现、但开发人员会完全遗漏的其他问题。在升级过程中，请确保包含所有必需的人员，以保证最终的升级是有效且安全的。

- *定义升级对数据源潜在的影响*

  升级会以各种方式影响数据源。例如，你可能会发现你需要为数据库增加新的字段，或者你需要访问一个全新的数据源。数据源可能是来自传感器的输入或是其他非传统的信息，这些信息是应用程序现在需要用来使功能正常的数据。当你考虑一次升级的后果时，请思考应用程序可能需要的所有输入。

与旧的应用程序不同，在要求清单中你再也不能只考虑自己的应用程序，或者在运营环境中只考虑自己公司的需求。你的升级要求清单将需要考虑第三方库、API 和微服务的使用。此外，你现在需要考虑将应用程序运行的环境作为整个要求清单的一部分。

还有一点很重要，不要忘记用户。当用户依赖桌面端系统时，创建一个可控环境是很容易的，在这个环境中很容易考虑到用户的需求。现在，用户可能会在当地的餐馆中用智能手机使用你的应用程序。用户（希望不是司机）可能在乘坐轿车时使用应用程序，或者在搭乘区间车或公共汽车时使用应用程序。升级要求需要考虑应用于各种环境中的各种设备。

13.1.3 定义升级的临界点

升级会从许多方面影响应用程序，而不升级的风险会根据应用程序、平台和需求的不同而不同。升级的临界点会直接与不升级带来的风险成正比。当一个升级变得很关键时，它在你升级清单中的优先级就会提高。计划升级周期的一部分就涉及给升级优先级赋值，这样公司就会首先进行最高优先级的升级。否则，你无法确定升级过程是否真的会帮助公司减少风险。当处理优先级时，请考虑以下这些标准。

- *安全风险*
  安全升级通常会有很高的优先级，因为已有人证明某些方法可以破坏你的系统。但并不是所有的安全升级都代表着马上会面临风险，因为还没有已知的黑客利用这个缺陷发起攻击，或者攻击需要特定的条件，比如需要访问物理机器。当对安全升级的需求进行优先级排序时，要考虑这个安全问题给应用程序带来的风险大小。

- *潜在的破坏*
  在升级清单中，对导致应用程序或数据损坏的问题进行修复的升级的优先级通常会比较高。根据风险的不同，修复问题的升级在很多情况下可以取代安全升级。在这种情况下，你必须确定这种损害发生的可能性以及导致的后果。

- *潜在的错误*
  由用户输入或其他问题制造的错误会带来巨大的风险，但是你能用额外的用户培训或通过新的公司政策来解决许多这样的错误。尽管如此，你必须基于错误发生的概率、错误可能导致的破坏以及公司解决、捕获和修正错误的能力来给出优先级。

- *自然事件*
  有一些升级会解决可能导致应用程序错误的自然事件。线噪声的存在会导致数据流中单个位的错误，所以一个解决方案是包含纠错码（error-correcting code ，ECC）程序来根除并修复该错误。在大部分情况下，随着能自动修复这些严重错误的技术的出现，你可以赋予这些问题较低的优先级。你仍然必须进行修复它们的升级，但是对肯定会导致问题的安全问题的升级必须首先进行。

给你的优先级列表赋值是很重要的。这些值有助于更容易地确定一个变更是否是关键的（因为它修复了一个安全漏洞），或只是为了方便（因为它做了用户认为比可用性更合适和完美的界面改变）。图 13-2 展示了当使用 5 个级别的优先级系统（你的系统可能包含不同的级别数量）时，不同的优先级标准之间可能的相互作用。

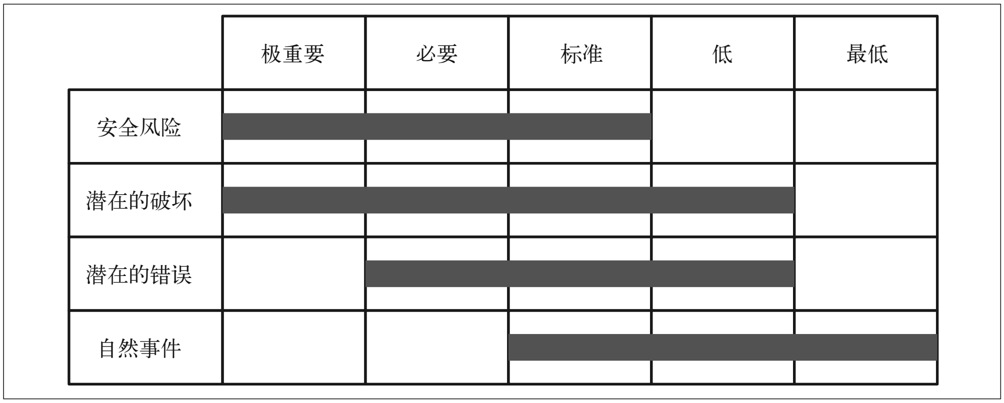


图 13-2：平衡优先级是测试过程中的一个重要部分

很重要的是要定期更新应用程序所需更新的优先级。当有新的变更要求时，要相应地提升现有升级的优先级。否则，新增的变更需求可能会降低一些现有升级的优先级。为了让列出来的升级能够完成其所要达到的目标，你必须持续更新。

13.1.4　检查升级的问题

任何时候你对现有产品进行升级，都可能会有兼容性或其他问题发生，而直到将升级迁移到生产环境之后你才会发现它们。到那个时候，要对问题进行处理已经太晚了，只能做出响应并期望最好的结果。一个更好的方案是在测试阶段查找潜在的升级问题，这意味着要有大量的测试团队或依靠第三方测试商（详见第 12 章）。但是，甚至在你进入测试阶段之前（在升级以任何方式被实施之前），也可以运用各种技巧来发现升级是否会制造问题。以下列出了一些较为常见的技巧。

- *查看供应商网站*
  供应商常常会提供关于它们产品已知问题的详细情况，包括兼容性问题。事实上，有时候为了推动产品发展，供应商可能会专门引入破坏性的变更，这肯定会导致你的应用程序出现问题。一旦认识到这些问题，你就需要考虑为了获得新特性而升级，进而对安全性和可靠性造成影响，这是否值得。
- *查看产品或供应商的论坛*
  在某些情况下，供应商会没有意识到有破坏性的改变或不愿意承认它。其他的用户会告诉你潜在的破坏性变更。这些不断弹出的怪异的错误信息在告诉你潜在的软件问题。当你开始关注论坛消息后，会对升级做进一步的调查，以避免在升级应用程序时遭遇别人遇到的问题。
- *阅读行业杂志*
  最糟糕的破坏性变更通常出现在行业杂志的某个地方。同时是作家的专业开发人员通常会查找那些使得产品几乎不能正确使用的产品变更。在某些情况下，你甚至能找到问题的解决方法。当一个破坏性的变更出现在行业杂志上时，你可以确定供应商们正在修复这个问题，所以比起立即进行更新来说，更好的方式可能是等待修复发布。

- 直接对新的第三方代码进行测试

  测试框架和测试脚本的使用（见第 11 章）是查找潜在问题的关键部分。测试可能不会发现所有的问题，但当一个问题足够大时，你可以确信测试会给你一个关于该问题的提示。

- 开发功能子集的测试场景

  如果你怀疑在代码中有破坏性的变更，但又无法找到关于它的明确的信息来源，那么可以尝试创建一个小型的聚焦于具体功能的应用程序。如果你真的发现有问题，那么创建一个只运行部分库、API 或微服务以进行验证的测试应用程序不会花费太多时间并且相当有用。测试应用程序可以作为开发解决方案或检查供应商何时修复问题的手段。

- 获取第三方的专业建议

  一些测试公司和许多顾问可以对库、API 和微服务提供专业建议，因为他们一直在尝试破坏这些代码。把工作时间都用来捶打代码的人肯定能提供一些优势，这些优势是那些很少写代码开发应用程序的人所不能提供的。

> 许多公司决定引入大量的变更作为一个单一的综合升级包的一部分，他们认为单次升级会比多次升级好得多。当所有的事情都完全按计划进行时，单次升级确实会有较小的破坏性。但事实是，事情很少按计划进行，而在单次包中进行多个升级会使得查找出错的过程变得复杂。使用阶段性的方法来引入变更通常是更好的方式。在生产环境中通常会出现你在测试环境中无法发现的问题，所以阶段性的变更可以帮你快速定位新的问题，这样你就能在它们成为大问题之前解决它们。

任何第三方代码的更新都不可能是完全没问题的。任何代码的变更都可能引入 bug、破解代码的变化、界面的改变、设计策略的变化以及各种其他无法简单避免的问题，因为变更像一枚硬币，有一面是好的，而另一面是坏的。抛硬币通常决定了你最终得到好的一面还是坏的一面。（当然，硬币也会有一个边，而有一些变更也确实是中立的。）

某些时候，你需要确定你找到的问题是否会导致如此多的问题，是否值得花费时间去升级。当你确定不值得花时间进行升级时，还必须考虑依赖的问题。你拒绝的代码可能会以某种方式影响一些其他的升级代码或应用程序，在没有分析的情况下你无法预见这些情况。花时间确定破坏性问题是否真的会导致在不升级时带来太多痛苦是值得的。取消应用升级并等待修复更容易释放出有缺陷的软件，从而使得用户愤怒并可能导致安全问题。

13.1.5 创建测试场景

测试一个升级要遵从很多与第 11 章和第 12 章所述相同的过程。但是，升级过程要求你执行一些强制测试以确保升级按预期工作。测试用例的使用可以确保在发布之前对升级进行完全测试。下面列出了在测试升级的过程中必须包含的一些测试场景。

- 故障单内容

  你进行升级所要解决的故障单应该包含足够的信息以创建测试用例。当情况不是如此时，你需要与负责研究修复问题的开发人员合作。在应用程序中修复的每个 bug（无论是编码错误还是其他非编码错误）都要求有一个测试用例来确保你完整测试了变更的代码。

- *行动项*

  有时候一个问题不会产生故障单，因为你无法重现该问题或因为其他原因（比如不能访问代码）无法修复它。尽管你只是怀疑一个问题，但尝试一个测试用例来找到它是很重要的。代码的变更通常会让一个间歇性的问题变得更加明显，从而带来了修复它的可能。但是，你只能在测试过程中发现这些机会。

- *第三方代码变更*

  对于每一个第三方代码的变更，你都应该为应用程序创建一个测试用例。你需要明确第三方代码的更改不会给应用程序带来负面影响。遗憾的是，你通常在集成测试阶段才发现这些问题，除非你有很棒的测试框架和一些检查第三方代码的测试脚本。

- *环境和配置变更*

  各种环境和配置的变更都会导致问题。例如，公司的政策可能会改变 IT 赋予用户访问资源的权限。这个变更是正面的，因为它保证了资源的安全并且减少了对资源的访问，所以应用程序可以运行得更快。但是，缺少访问权限也会给用户带来麻烦，所以你需要确保用户仍然能够访问他们需要的资源。其他类型的环境及配置的变更会对应用程序有类似的影响，并且直到你进行了必要的测试之后才能知道这一情况。

13.1.6　实施变更

此时，你已经考虑了一次升级的所有元素。所以，这项工作的关键是要确保升级可以：

- 按期般工作
- 保持稳定
- 提供适当的安全性
- 提升可靠性
- 创建很棒的 UI
- 堵住任何安全漏洞

实施过程应该首先从清单中最高优先级的项目开始，而不是尝试一次实现所有项目。许多公司面临的一个问题是升级变成压倒一切。变更常常会发生，但是，是在经过测试和满足规范要求的情况下进行的。因此，升级本身最终需要一次升级。某些时候，代码会变得非常混乱，没有人能够完全理解，并且没有人能够保证安全。你得到的会是黑客最喜欢的软件，因为很容易攻克任何建立在其上的防御措施，并且也很容易隐藏攻击行为。

13.2　制订升级测试计划

有时候，你需要为升级编写代码。你不应该一直等待这个过程结束才开始测试。要尽可能快地开始测试，特别是当与第三方测试商合作时。你越早发现潜在的问题，修复它们所需的成本和时间花销就越少。考虑使用 mocking，这样你就能在功能可用时进行升级测试，而不用因为依赖而等待整个升级。为此，接下来开始为升级制订并实施一个测试计划。

13.2.1 执行所需的预测试

一旦升级过程开始，你就需要立即开始测试过程。许多人不明白贸然是什么意思，但大部分公司还没解决问题就把钱花光了。为了确保测试能够恰当地进行，你需要立刻开始测试。在升级代码的过程中，你必须执行以下这些级别的测试。

- 单元

  确保你完成编码时测试了每一个变更。否则，你无法确定代码是否可以工作。使用 mocking 有助于在这个级别上使测试过程变得可行。

- 依赖

  你做出的变更会影响其他代码。为此，需要测试相关代码里的依赖项，这样你就有机会以轻松的方式进行集成测试。

- 配置

  确保对配置的设置仍然能够提供 IT 专家期望的效果和灵活性水平，这是很重要的。否则，当你以不同的方式开始配置时，应用程序会在集成测试阶段失败。

- 安全性

  你不能简单地假设你对代码做的修复真的会像预期那样工作。有时候安全修复并不能真的修复问题。你需要对未能确定是否真的堵住了安全漏洞的修复代码进行测试。此外，你必须确保修复不会引入新的漏洞。



有些开发人员认为安全测试是查找代码中的漏洞。是的，这是一种安全测试。但是，你必须测试安全性的所有方面。例如，一个通常被遗漏的安全测试级别是确保用户最终有合适的角色并且有所需的访问级别。执行用户级别任务的管理者应该是一个用户角色，而不是管理角色。黑客会利用角色问题来用比本应该的授权更高的授权级别入侵系统。

- 数据连通性

  应用程序管理数据，否则就没有理由开发应用程序了。你的测试必须确保测试的单元仍然能够连接到所需的数据源并成功管理它们。有时候，我们会很惊奇地发现对安全问题或其他问题的修复实际上最终会使得数据不可访问或者破坏应用程序安全管理数据的能力。

13.2.2 执行所需的集成测试

当你把升级放在一起时，就需要查找集成问题。因为你不是从零开始写代码的，所以要找出集成问题会比较麻烦。你要把最近大家写的新代码和很久没有人看过的旧代码混合在一起。当专门开发新代码时，开发人员理解集成问题的能力是比较弱的。因此，你需要执行第 11 章所描述的那种集成测试，但是要关注新旧代码之间的接口。

如果有处理过旧代码且完全理解它的人，对查找集成问题是很有帮助的。考虑到人们会不断变换工作，你可能不得不要寻找几个适合的人并聘请他们作为顾问来解决出现在旧代码中的问题。聘请熟悉旧代码的人可以为指出旧代码的问题节省大量时间和工作量。此外，在之前的开发团队中工作的人员可以解释为什么有些代码要这样工作，以及为什么团队要以某种方式处理事情。文档可能会提供这类信息，但文档通常是不充分的且没有良好的组织，所以有时候它带来的问题比答案更多。

请用现有的测试框架和脚本来进行测试，以确保应用程序仍然能满足基本要求。你可能必须对测试框架和脚本进行更新，以确保你能为新特性提供测试功能并且移除对废弃特性的测试。你要对测试框架和脚本所做的变更维持一份变更日志。否则，当某一个升级被证明是不正确的时候，你无法回退变更。

13.3 将升级移到生产环境

一旦测试完成并且确定升级可以正常工作，你就需要将其迁移到生产环境。如果你的公司很大，尝试首先对部分用户进行升级，这样如果这些用户发现了错误，你可以在不影响整个公司的情况下修复它。重要的是要记住应用程序必须在升级时保证可靠和安全，还要满足用户要求的速度和界面的体验。否则，升级会因为用户拒绝使用而失败。他们会想要使用感觉更舒服的版本，即使那个版本可能会导致数据泄露或其他安全问题。

你要聘请用户作为测试人员的部分理由就是要确保他们会告诉其他人关于新升级的信息。当用户对升级感到兴奋时，你会得到更好的响应，并且会花更少的时间来让每个人都开始使用新产品。当然，你要确保测试人员有比较好的经验。

从安全的角度来说，聘用第三方测试商对升级进行入侵测试是有帮助的。是的，大部分人会说你应该在测试服务器上完成所有必需的测试，就绝大多数情况而言他们是正确的。但是，入侵测试不仅仅检查应用程序的错误，它还提供了对以下这些项目的完整检查。

- 网络连接
- 入侵检测软件
- 入侵防御软件
- DMZ 可靠性
- 所有硬件需要的固件更新
- 服务器设置
- 所有软件所需的更新
- 应用程序环境
- 第三方库、API 和微服务
- 应用程序
- 用户进程

- 数据库连通性

通过测试不太可能检查所有这些项。例如，你可以通过测试来检查生产服务器遭受攻击的脆弱程度。虽然看起来有些测试问题不属于迁移升级到生产环境这个范围，但是对服务器设置（包括应用软件）的任何变更，都确实需要这样一种测试。不同组件之间的交互使得你无法知道在变更之后服务器是否仍然可靠和安全。遗憾的是，许多公司并没有做好这方面的工作，因为似乎没有人理解服务器设置发生了什么。关键是服务器现在可能很脆弱，而你可能完全不知道。

测试软件不是你要考虑的唯一事情。获取用户对界面改变的反馈也是很重要的。用户每天都有很多时间在细致地使用应用程序。用户很可能会比开发人员更早遇到由于界面改变而导致的潜在安全问题。事实上，经验表明，开发人员经常发现不了具体变更导致的问题，而这些问题对于用户来说是非常明显的。这并不是说开发人员不够敏感或没有正确地完成工作，只是因为用户对于应用程序应该如何工作有完全不同的视角。

对于一次升级来说，检查升级是否会影响管理员、开发运营人员、DBA 和其他技术人员也是要优先做的事情。你需要确定变更是否会给用户带来负面影响。例如，在系统中花费太长时间传播的配置更改代表着潜在的安全漏洞。如果用户的删除操作需要花 24 小时才能生效，愤怒的用户会导致各种难以跟踪的问题。所有这一切都是会导致公司发生各种各样悲剧的问题，所以确保公司的 IT 专家能够完成他们的工作是升级很关键的一部分。

第14章

考虑更新选项

更新意味着将新的信息、特性或界面元素带给应用程序。提高应用程序准确性也属于更新。一次更新可能不会显著影响应用程序的代码，但仍然能够以多种方式影响安全性。例如，更新数据库使其包含一个字段，用于显示谁最后编辑了该记录，这就可以跟踪已经发生在系统中的错误或病毒感染的源头，从而提高安全性。更改提示以使从用户那里得到的信息更加清晰，是一种不需要任何重新编码的安全修复，特别是当这些提示存在于外部文件中时。更新的本质是使应用程序在某些方面变得更好，从而可能不再需要代码修复了。重要的是要知道何时区分升级和更新，所以本章首先会比较这两个过程以及它们如何影响应用程序。

更新不总是能修复安全问题，就如升级有时候是多余的。为了有效利用公司资源，你需要知道何时进行一次更新以及何时确实需要进行一次升级。第 13 章描述了升级过程并描述了这一过程的一些细节。本章会比较更新与升级，并帮助你理解更新过程是如何工作的。

更新可以分为几类。例如，你可以选择更新编程语言套件。新的套件可以提供更好的调试、经过优化的代码输出以及更快的可执行文件。这些特性没有一个会直接影响应用程序代码，但它们确实能影响应用程序的工作。只要重新编译应用程序模块就能让所有人看到它的效果。

在某些情况下，你需要进行紧急更新。推行更新管理是很困难的，因为更新常常根本不会影响应用程序代码。在不改变任何一行代码的情况下，你很难告诉管理层你需要时间、资源和人力来进行一次更新。在你真正进行了更新并以量化投资回报的方式向管理层展示了更新的效果之前，尝试展示更新的好处也是很难的。

与升级一样，你需要测试更新。但是，某些更新测试与升级测试有很大的不同，因为进行更新时不必处理代码。总之，本章会针对这种类型的更新给你完整的入门引导，使你可以更好地判断何时使用更新和升级。

14.1 区分升级和更新

升级和更新都会影响应用程序的安全性。但是，升级与更新在范围和目标上有很大的不同，所以区分两者的不同是很重要的。比起更新来说，大部分升级会创建重要的新功能并在较深层次上影响代码基础。事实上，许多更新完全不会触及代码。更新会执行诸如以下这些任务。

- 更改应用程序管理的数据库。例如，一个提供目录信息的数据库可能会有一个对描述的更新，以帮助用户作出更好的选择。
- 修改现有特性使其变得更友好或不易出错。例如，菜单输入现在可能包含快捷键，这样用户就不必去猜要使用哪些组合键。
- 重新配置 UI 特性，使它们更好地满足用户的需求。更新可能会包含对更多语言的支持，这样用户就能用他们熟悉的语言来使用界面，而不是他们不太熟悉的第二或第三语言。
- 包含更多特定的 UI 选项，比如将文本输入框换成单选按钮，这样用户就能作出特定的选择。每一个从应用程序中移除的元素都自然地会提升应用程序的准确度和安全性。

在大部分情况下，从开发人员的角度来说，更新是比较微小的。在某些情况下，开发人员很可能不需要做改动。例如，你需要 DBA 提供服务去更新数据库，而不是依靠开发人员来做这件事。DBA 在大部分情况下可以更快和更精准地执行这项任务。设计师可以使用当今可用的一些编码技术来作出对 UI 的更改，从而使开发人员完全不需要更改代码。但是，当要对更新进行测试以确定其是否像预期那样执行时，开发人员要参与其中。有时候更新也需要一些代码调整以使其能正常工作。

即使开发人员没有明显参与到更新过程中，相关团队也应该在讨论时包含开发人员。事实上，更新与升级一样，应该得到开发团队中所有利益相关者的全力支持。这个过程要包含用户测试者，他们会验证更新是否真的按要求进行，并确定更新没有导致明显的新的用户错误或其他潜在问题（比如使用户的操作减慢或使培训成本激增）。

升级和更新普遍要做的一件事是测试。某些公司错误地认为更新需要较少测试，或可能完全不需要测试。这种观点带来的问题是，将更新放到生产服务器上很可能会出问题，而测试本可以更早地发现这些问题并更容易进行修复。重要的是，要记住测试不仅仅是检查代码，它还会检查诸如用户在某些级别上可能无法与应用程序交互这样的问题。更新实际上可能会产生新的安全漏洞，使得黑客能够以新的方式（特别是当更新涉及新的 UI 功能时）与应用程序交互。但是，更新测试的重点在于被更改的区域。例如，你仍然要测试应用程序的每个方面，但如果 UI 发生了改变，测试的重点就要放在 UI 上，并且你要在这个方面做更多的检查。

某些更新会超出开发团队直接监管的范围。例如，库、API 或微服务的第三方更新可能会自动发生。在这种情况下，你必须进行测试，但是大部分情况下测试实际发生在生产系统接受了更新之后。回退更新估计也是不可能的，所以破坏性的变更可能导致公司寻求新的第三方产品或进行修复该问题的更新。因此，关键是要保持跟踪各种更新并准备在任何时候测试它们。

14.2 确定何时更新

第三方软件在进行更新时可能会出现问题。当修复可能使供应商面临尴尬时，不少第三方供应商会尝试悄悄地进行而不通知开发团队的更新。安全更新有时候属于这种类型，它们会在某天神奇地出现而没有人能真正知道第三方供应商何时发布了它们。这些更新会滚雪球一样地累计并导致应用程序停止工作，或者更糟糕的是，开始以不可预测的方式工作。

你对软件进行的更新或者第三方供应商为你做的更新，都需要一定程度的计划，即使当第三方供应商忘记提及更新或忘记为什么要更新时。你需要知道一些变更实际上是属于更新而不是升级。为此，接下来会讨论一些在确定应用程序是否需要更新时应当考虑的问题。

14.2.1 处理库的更新

库的更新是最困难的，因为库会成为应用程序代码的一部分。因为你直接将其引入到应用程序中，所以有些变化难以隐藏。此外，某些变更会导致奇怪问题的出现，那些 bug 一直都在库里面，但从来没有展现出来，因为应用程序以不同的方式分配内存或其他资源。在包含新库之后，更新甚至可能不会出现在应用程序中，因为调用可能有稍微不同的名称。对于你想进行的各种库的更新来说，这些问题都是很普遍的。接下来会将库分为从第三方集成的和自己开发的这两种类别。

1. 处理第三方库的更新

当处理第三方更新时，第一步是要获取变更清单，假设供应商提供了变更清单。根据这份清单来确定哪些变更确实会影响到应用程序。在你制订计划进行变更之前，使用单元测试来帮忙确定文档中写的变更是否能够真实反映函数中的变化。

遗憾的是，这份清单不会告诉你一些问题，比如改变名称不会以变更的形式出现。在许多情况下，供应商会将名称的变更作为一种新特性。因此，你还必须检查新特性列表，查找你已使用的函数变更后实现的特性。例如，新特性使用名称 `MySpecialFunction2()` 替代了 `MySpecialFunction()`。这个变化能够反映出新特性，这意味着你必须仔细检查该更新，以确保你理解其对代码造成的结果。

如果你能获得代码的副本，可以使用静态测试技术来更好地理解供应商对库做出的变更。通过调试器来运行代码，看看代码的行为也是很有帮助的。在许多情况下你可能没有这些选项可选，但考虑一下它们并没有什么坏处。

2. 处理内部库的更新

执行内部库的更新可以让你对更新过程有更多的控制。你可以执行各种级别的测试，因为你自己拥有代码。最重要的就是能用文档完整记录更新引起的每一种更改，这样你在后面就不会遇到可靠性和安全性问题。当更新改变了函数的工作方式而代码继续用旧的方法来使用函数时，许多安全问题就会发生。这种不匹配的情况会导致安全漏洞。

一种更糟糕的情况出现在更新对函数进行了重命名，却缺少文档从而导致使用这个库的人员继续使用旧的函数名。其结果是开发这个库的团队知道这个库包含了一个安全修复，但开发应用程序的团队并不知道这个修复的存在。黑客事实上会在可能的时候检查这种库的变更并利用这种不匹配来发现安全问题。利用不匹配的情况会比真正寻找编码错误更容

易，因为重命名的函数就像灯塔广告一样告诉黑客有修复存在。

请使用静态测试来确保文档中的变更确实能匹配库里面的代码变更。在进行了静态测试之后，要使用单元测试来验证这些修复是否能正常工作。14.5 节提供了关于测试过程的额外信息。

> 最糟糕的安全漏洞是所有人都认为进行了某项修复，但其实修复并不存在、无法正常工作或者开发人员没有在应用程序中实现它。要以持续的方式验证开发人员确实实施了更新。即使一个修复无法正常工作，持续的实施可以让接下来的更新更加容易实现，因为每个人都在以相同的方式使用函数。

14.2.2 处理API和微服务的更新

API 和微服务都会遇到同样的问题，即它们在大部分情况下都像黑盒一样工作。有可能在没有任何人知道的情况下进行一次更新。某些供应商在过去确实是这么干的，并且在处理内部代码的时候也发生过这种情况。当 API 或微服务无法正常工作时，黑盒会导致各种问题。要追踪责任人是很困难的。为此，很重要的是要确保开发团队对 API 或微服务调用的内部运作的每次变更都进行文档记录，并且要保证对于给定的输入要得到相同的输出。接下来会讨论第三方与内部更新的区别。

> 应用开发人员会把对 API 或微服务的信任建立在这样的基础之上，就是能够保证对于一个给定的输入，其输出能够保持相同。一次更新可以使调用更加安全、快速或可靠，但它不能改变输出。如果一次更新改变了一个接口的任何元素，那你需要创建新的调用来访问这个元素。这个方法确保开发人员为了获得更新带来的好处，将需要发起新的调用。

1. 处理第三方API和微服务的更新

当处理第三方 API 和微服务代码时，你无法访问任何更新代码且无法进行任何类型的静态测试。你可以做的就是执行密集型单元测试来确保以下这些考虑事项是真的。

- 对于给定的输入，每一个调用都能产生在原先指定范围内的正常输出。
- 调用不能接收原先指定范围外的输入。
- 任何涉及利用代码（确保安全措施能够工作）的测试都要失败。
- 速度测试显示调用能够在原来的时间容许范围内，或者更快地成功完成任务。
- 可靠性测试显示调用不会在高负载下失败。
- 任何文档中记录的对代码的变更确实能正常工作。

2. 进行内部API和微服务的更新

对于内部更新，总是要将进行静态测试作为首要的测试方法。在这个过程中，你要验证文档与代码的变更能够完全匹配。更新中的一个重要问题是 API 或微服务的开发团队认为更新会以某种方式工作，但它实际上却以另外的方式运行。将文档中的更新与故障单进行匹配以确保修复解决了故障单中的问题也是一个好的做法。这种可见的检查过程可能看起来

很花时间，但执行它通常会使后续的测试更加容易，并且能够降低安全问题悄悄潜入代码中的可能性。API 和微服务代码的黑盒特性使得对它们进行静态测试甚至比库更加重要。

内部代码也会有益于某些第三方代码的检查（详见 14.2.2 节）。但是，你现在可以对手里的代码执行这些检查以更好地理解为什么有些检查会失败。

处理微服务更新时的一个棘手问题是许多公司现在依靠微服务 API 来使各种微服务看起来像是一个自定义的 API。微服务 API 的使用可以降低开发人员的编程困难，并确保每个平台通过使用提供给该平台的协议拥有所需的资源。图 14-1 展示了一个典型的微服务 API 架构。

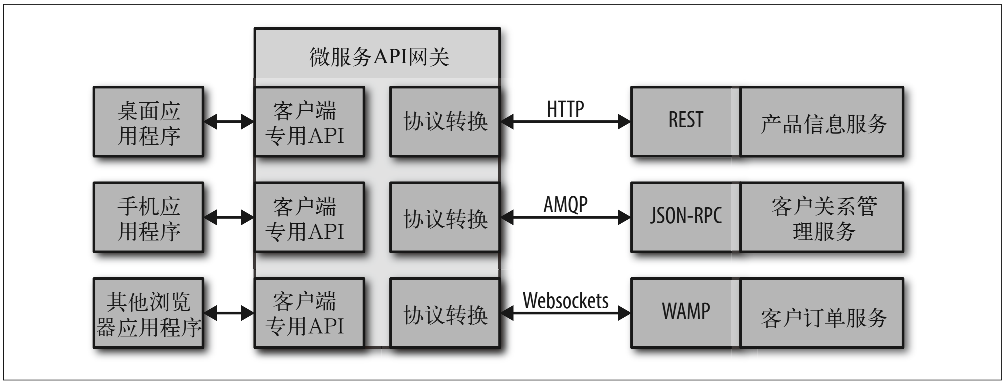


图 14-1：当使用微服务 API 时，也要测试 API

直接测试各种微服务以确保更新没有改变微服务原生的工作方式是很关键的。但是，你还必须测试微服务 API 是否仍然能正确地响应应用程序支持的所有平台类型。否则，你无法确定微服务 API 会继续以安全和可靠的方式提供正确的结果。

14.2.3 接受自动更新

某些供应商现在令忽略自动更新变得很难。自动更新是供应商以某种方式推送到客户端的更新。在某些情况下，供应商简单地在服务器上变更软件，这样你最终会自动使用更新。以下是供应商喜欢自动更新的主要理由。

- 减少由于使用过时版本产生的安全性和可靠性问题
- 减少支持成本
- 较少的人工需求
- 提升投资回报率（ROI）

当然，问题不在于自动更新是否对供应商有好处，而在于它们是否对你的公司有好处。在许多情况下，使用自动更新证明是有问题的，因为公司并没有准备好更新且破坏性的变更通常会导致支持问题。最常见的几种情况会导致数据丢失或安全漏洞，这与最开始进行更新的目的相悖。

为了减小产生意料之外的副作用的几率，大部分公司都尝试在可能的时候避免自动更新。当第三方供应商坚持自动更新而你没有这个第三方软件的替代产品时，那你真的需要有人

跟进第三方软件以确定自动更新将在何时发生。立刻测试更新可以帮你避免破坏性更新的问题，并且也可以潜在地提升公司应对突发变更的能力。

14.3 更新语言套件

Web 应用程序属于这样的一种类别，使用任何一种语言的用户都能在搜索其他东西的时候发现它们。你开发的任何公开的应用程序，即使你是专为某一个团体的使用而开发的，也是处在互联网的世界舞台上，来自任何地方的任何人都能够访问它。理论上，你可以尝试阻挡访问，但这是一个会起到反作用的策略。如果你突然发现应用程序正在接受来自你从来没有想象到的地方的赞美，你可能需要一次应用程序更新来解决与语言相关的问题。接下来会讨论某些你需要考虑的问题，特别是在要维护一个安全的应用环境时。

14.3.1 创建语言支持清单

不是每一个应用程序都会支持世界上的所有语言。尝试这样做会是一个逻辑噩梦，并且也不需要，因为不是所有人都会真的想要使用你的应用程序。但是，当应用程序变得较为热门时，你可能会发现你需要支持不止一种语言。现代的服务器软件通常可以跟踪访问应用程序的位置。位置本身可能不会提供你所需要所有东西。例如，你可能会惊讶地发现罗马尼亚有很多官方语言，如下所示。

- 罗马尼亚语
- 克罗地亚语
- 德语
- 匈牙利语
- 罗马尼语
- 俄语
- 塞尔维亚语
- 斯洛伐克语
- 乌克兰语

然而，这些只是官方语言。你的应用程序的用户可能实际上使用着一种不同的语言，比如意大利语。不可能仅仅基于位置就能准确知道要支持哪种语言。关键是如果你突然发现应用程序正在收到大量来自罗马尼亚的请求，你可以意识到用户的母语不太可能是英语（或者可能根本不会讲英语）。如果你的应用程序只支持英语，那么你可能会发现用户会迷惑、犯错并且可能会导致安全和可靠性问题。



在某些情况下，你可以根据应用程序的环境来假设要支持的语言。例如，一个在美国的私有环境中运行的应用程序，可能只需要支持英语和西班牙语。当应用程序在罗马尼亚运行时，你可能要支持所有的官方语言以确保每个人都能访问你的应用程序。考虑应用程序环境是创建语言支持清单的一个重要部分。

使用位置信息会为你提供一个良好的开端，但你需要以某些其他方式获取语言统计信息。确保你能获得良好数据的唯一有效方式大概就是提供反馈表单或进行用户调查。当计算关于应用程序使用的统计数据时，你会看到其浮现出来的模式并可以开始确定要支持哪些语言。创建一个支持语言的清单是很重要的，因为你需要知道哪些语言你真的可以作为更新的一部分提供，并且你要确保你的首要目标是那些最常用的语言。

当要进行一次更新以支持多种语言时，你通常要为每种语言提供不同的 URL。否则，一个人几乎不可能找到他所需要的语言。此外，使用不同的 URL 可以检测浏览器提供的信息并为用户作出合理的语言选择。下面是两种常见的为每种语言创建唯一 URL 的方法。

- 与语言相关的 URL
  相同的域名支持不同的语言。比如一个 http://www.mysite.com/en/stuff.html 这样的 URL 会支持英语，而 http://www.mysite.com/de/stuff.html 则会支持德语。这种方法的好处是你不需要购买多个顶级域名（top-level domains，TLD），从而可以省下一笔钱。这种方法的关注点在语言。
- 多个顶级域名
  每一种语言都有它自己的域名。一个像 http://www.mysite.us/stuff.html 这样的 URL 支持在美国的用户，而 http://www.mysite.ro/stuff.html 会支持在罗马尼亚的用户。这种方法的好处是，搜索引擎（比如 Google）会为每一种语言提供一个单独的链接，而不是将所有语言都作为重复的内容且只显示一个链接。这个方法的关注点在国家 / 地区。你可以在 Wikipedia（https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains）上找到 TLD 列表。

14.3.2 获得可靠的语言专家

如果你选择对应用程序进行语言相关的更新，只有当更新看起来就像是你用目标语言重写了原来的内容时，你才能保证它可以可靠和安全地工作。每个人都处理过这种指令或其他的书面交流，就是某人原来以一种语言书写，然后翻译成你所讲的语言。当翻译器执行得不如普通水平时，结果常常是很滑稽的。各种信息背后的真正意义会丢失且没人能够重视信息。但是，这种误解导致的安全问题非同小可。

然而，如果你无法提供进行翻译的正确环境，即使是最棒的翻译人员也做不出最好的工作。以下步骤会带你深入了解一些你能用于创建更好的应用程序翻译的技术，并因此确保用户真的能遵从你的引导。

(1) 将所有在应用程序中硬编码的字符串放置在某种类型的数据库中，通过在应用程序中的位置来标识每个字符串。
(2) 将每个硬编码字符串用占位符控制替换，当页面加载时你可以用正确语言的字符串填充。
(3) 为每一种语言创建一个单独的数据库文件并确保数据库名称能反映语言内容。
(4) 用翻译人员提供的翻译后的字符串填充每个语言数据库。为每一个字符串准确地使用相同的标识符，这样该标识符总是能匹配屏幕提示。

(5) 给应用程序添加代码以执行任务，用来自数据库的正确提示符填充占位符。
(6) 开始用原来的语言测试应用程序以确保提示符能正常工作。

> 对语言更新进行二次检查通常是一个好主意，特别是当公司里没有人能地道地讲那门语言的时候。在进行翻译时通常会有潜在的错误。此外，你需要考虑讲这门语言的人的社会习俗。最棘手的问题在于某些语言对于一些词没有直接的翻译。因此，你必须选择正确的近义词，然后确保不会对近似翻译产生误解。对翻译进行独立的检查，除了能够通过降低输入错误的可能性而使应用程序更安全之外，还会避免以后的窘迫。

14.3.3 验证与语言相关的提示符能否在应用程序中起效

简单地翻译提示符在多数情况下是不够的。你需要依靠语言专家来确保屏幕上的每个提示都能反映出母语者期望如何看到这些提示。例如，有些语言是从左向右读的，而有些是从右向左读的。提示的方向是很重要的，正如格式化一样。你可能需要提供应用程序提示及格式来确保使用应用程序的用户能真正正确理解提示。

14.3.4 确保数据以正确的格式呈现

世界上的不同区域会用不同的方式来格式化数据。某些国家 / 地区会使用逗号而不是句号来表示小数。同样，这些国家 / 地区可能会使用句号而不是逗号来作为千位指示。想象一下在这种情况下，如果用户基于错误格式更改了数值会发生什么。用户可能会输入一些值导致应用程序错误执行、崩溃或只是做一些奇怪的事情。其结果可能会是一个安全漏洞或是黑客制造安全漏洞的开端。许多安全问题都产生自看起来愚蠢的错误，黑客可以利用其来获得很小的好处，但却会导致很大的损失。

以正确的形式展示数据可以确保不会产生误解。仅仅因为逗号和句号之间的混乱而造成潜在的安全问题，这看起来似乎是你能轻易解决的问题，但有时候却证明是异常困难的。你可以在网页 http://docs.oracle.com/cd/E19455-01/806-0169/overview-9/index.html 上找到关于各种语言的千位提示符和小数点的例子。在网页 http://www.statisticalconsultants.co.nz/blog/how-the-worldseparates-its-decimals.html 上的图形指示也很有趣。

14.3.5 定义语言支持测试的特殊要求

当你开始添加多种语言的时候，测试应用程序会变得困难得多。除非你计划让那些语言专家在应用程序的生命周期中一直提供支持，否则你需要一些其他方法来验证更新不会导致语言支持问题。合适的计划、变更文档以及用原来的语言进行验证是最佳的开端。任何时候，如果你对应用程序进行的变更会影响原有语言的提示，那你必须考虑也要改变所有其他支持的语言的提示。要使这个过程尽可能简单，请采用以下这些建议。

- 计划
  确保你仔细检查了每一处改变。有时候设计师会尝试调整这些提示，而实际上并不需要做出任何改变。尽可能减少变更数量会降低成本，确保应用程序保持可靠，并能通过减少失误来帮忙提升安全性。

- 文档

  对原有语言的每个提示的改变都必须同步到应用程序支持的其他每一种语言中。当语言版本不同步时，安全和可靠问题也会开始出现。更新过程的这一部分能够顺利无误地运行是很关键的，否则你又需要聘请语言专家来处理混乱。

- 验证

  很难确保每一个改变都确实发生了，因为你可能没有会使用你所要支持的语言的员工。验证语言更新的一个好方法是使用屏幕快照。比较原来的屏幕和更新后的屏幕来验证提示真的发生了改变。这种方法的唯一问题在于你无法保证改变的提示是完全正确的。

由于应用程序使用提示的方式，你无需测试每一种语言的底层代码。你对每一种语言做的变更是在界面上，而这也是你测试与语言相关的问题要关注的地方。确保你能获得好的提示且那些提示出现在正确的地方是语言支持测试的基础。

14.4 执行紧急更新

升级通常是精心策划和排期的。大部分升级也会有合适的计划和时间安排。但是当黑客刚刚让你的应用程序暴露了一些需要立即响应的可怕漏洞时，有时候更新无法等待，此时你必须创建一个临时的修复并立刻发布它，否则就要承担后果。没有人喜欢紧急情况。应用程序越关键且受影响的数据越敏感，人们就越不喜欢紧急情况。接下来的几小节内容有助于处理在深夜潜伏着、等待着你的防卫下降时出现的紧急情况。

14.4.1 尽可能避免紧急情况

每个人在应用程序支持过程中的某个节点上都会遇到紧急情况。这是很正常的。但是提前做好计划，你可以避免大部分的紧急情况。以下是在避免紧急情况发生时需要考虑的一些问题。

- 在发布之前对每个更新和升级执行完整的测试。
- 只有当系统负载和能力可以支持的时候才发布更新和升级。
- 避免在只有低级别员工在场或关键人物缺席的情况下发布更新和升级。
- 如果可能，确保应用程序能够访问来自多种来源的所有必需的资源。
- 制订预留生产力的计划并确保你有足够的额外能力来满足紧急的需求。
- 假设黑客会在最糟糕的时间发起攻击并（尽可能地）提前为这个攻击做好计划。
- 定时测试系统的安全性和可靠性。

14.4.2 组建快速响应团队

你的公司需要一个快速响应团队，即一组随叫随到来处理紧急情况的人。这个团队不需要一直都是相同的人员，因为每个人都需要休息，但你应该一直都有这么一个可用的紧急响

应团队，特别是在那些黑客喜欢攻击的时间（比如周末）。一个紧急响应团队需要包含应用程序进行各种更新时需要的所有人员。如果你平时需要一名 DBA 来进行更新，那紧急响应团队就需要一名 DBA。这个团队应该为处理紧急情况一起参加专门的培训，以确保在紧急情况发生时他们都做好了准备。

> 永远不要将进行升级作为处理紧急情况的方法。升级会深深地影响代码，而且你无法在紧急情况下用很短的响应时间进行适当的测试。当你在未经适当测试的情况下发布升级时，很可能会包含黑客非常喜欢的某些 bug。你要始终稳扎稳打，仅将紧急措施用于可通过升级解决的问题。

14.4.3 执行简化的测试

根据紧急情况的特性，你可能必须在发布更新前进行少量测试。至少，你必须测试更新会直接影响的代码，可能还要测试其周边的代码。当黑客闯进来时，你可能没有时间像平常那样对代码的每个部分都进行测试。重要的是要记住你未发布的更新可能会牵制住黑客。这就归结到一个平衡风险的问题，一边是紧急情况而另一边是可能会制造出更糟糕的问题。

14.4.4 制订持久的更新计划

永远要将紧急更新作为临时的方案。你无法知道快速响应团队在发布更新时会是多么匆忙。在这个过程中整个团队可能会制造各种错误。即使更新生效了，你也不能放任紧急更新不管，不进行后续的审查。放任紧急更新不管会打开你可能并不知道的安全性和可靠性漏洞，并可能在将来给你造成相当大的问题。

> 事后对紧急情况进行走查，确定是什么原因导致了它，并考虑快速响应团队是如何响应的是一个好的做法。通过走查，你可以总结经验教训以便将来处理紧急情况。黑客肯定会持续提升他们的技能水平，所以你也必须提升技能水平。花时间理解紧急情况并寻找更好的处理方法是组建一个更好的快速响应团队的有价值的方式。

作为持久更新过程的一部分，请确保对更新进行静态测试，以验证快速响应团队作出了适当的响应并很好地修复了问题。确保尽快对持久更新做出计划，以保证在别人利用其他问题之前修复它们。新的更新应该依赖所有平常的测试过程并遵循你为持久更新制订的计划。

14.5 制订更新测试计划

无论你如何创建更新或时间安排，在没有测试的情况下匆忙发布更新是不值得的。匆忙发布更新最后几乎总是会导致问题。事实上，匆忙发布更新产生的问题通常比不进行更新带来的问题更糟糕。测试会迫使你慢下来，重新检查创建更新的思维过程。如果使用

正确，测试可以确保团队成员与你一样确定更新是可靠的，且不会导致最终成为安全噩梦的问题。

更新测试不会遵循跟升级测试完全一样的模式，因为你在更新期间对应用程序的改变要少一些，并且可以将大部分精力聚焦于被改变的特性。此外，你可能会在一次紧急情况中进行更新，并且需要尽快实施这些更新以防止出现重大的安全漏洞。为此，你必须执行以下这些级别的测试以确保更新不会弊大于利。

- 单元测试
- 集成测试
- 安全测试

当更新用于处理紧急情况时，这三个级别的测试通常可以防止更新弊大于利。但是，你必须尽快执行以下这些级别的测试。

- 可用性测试
- 完整性测试
- 访问性测试
- 性能测试

一旦事情尘埃落定且你有足够的时间进行测试，就需要确保这次更新是完全起作用的。确保功能性的最佳方式是执行以下这些级别的测试。

- 回归测试
- 国际化和本地化测试
- 一致性测试

你可以进行其他类型的测试，但只是在需要的时候。例如，Web 应用程序很少需要安装，所以在大部分情况下你不需要进行任何类型的安装测试。例外的情况是 Web 应用程序要求安装浏览器扩展、浏览器插件、浏览器服务、系统代理或其他必需软件时。

第 15 章

考虑报告的需要

报告有助于记录应用程序的状态。虽然似乎代码文件里的注释或在走廊里的简单讨论就足矣，但一个接收关于应用程序的信息的正式过程真的是确保每个人都能理解应用程序状态的唯一方式。许多人将这类报告与 bug 等负面问题关联起来。但是你可以为各种需求使用报告。例如，基于使用统计信息，一个正面的报告可能会表明用户确实喜欢某个新特性。类似这样的报告会给每个人以鼓励，而不用担心是否做了无用功。

你通常要为整个应用程序创建内部报告，但也可以使用报告来跟踪单独的应用特性。就这一点来说，你甚至可能不需要直接关注应用程序，而是关注网络带宽的使用或数据访问需求。重点是内部报告通常聚焦于你的代码、用户和资源。

外部报告来自第三方。例如，如果你聘请了第三方测试商（见第 12 章），那么在测试期间会希望看到大量关于应用程序状态详情的报告。当你要访问库、API 或微服务时，第三方报告还应该列出应用程序真正使用的服务。通过跟踪这些服务的实际使用情况，你或许可以优化应用程序或通过减少这些服务的订购来降低开支。

报告的一个重要形式是积极地寻求用户反馈（或者指定具体的用户代表来确保每类用户都有表达自己愿望的机会）。你永远无法满足应用程序的每一个用户。但是，你可以合理地寻求满足大部分用户需求的方法。关键是要确保你从用户那里获得的报告确实能够代表主要的观点，而不是少数人的不满发泄。让用户快乐、高效以及受控通常是很符合应用程序开发目标的。

> 报告可以以各种不寻常的方式给你的应用程序带来好处。例如，你可以通过分析报告来找出不寻常的使用或访问模式。不寻常的模式会表明黑客的活动，但也会表明应用程序升级（见第 13 章）或更新（见第 14 章）的需求。关键是要保持思想开放，当机会出现时能够以非标准的方式使用报告。

15.1 使用报告以做出改变

在你用报告做任何事之前，需要有一份报告。有些开发人员认为有一套标准的报告能满足每个公司的需求。遗憾的是，并没有。报告会因为需求的变化而变化。你需要仔细考虑如何创建你需要的那种报告并且避免只会浪费大家时间的无用报告。有时候你需要用报告来跟进一次升级或更新，以确保这些任务按计划进行。你可能会根据易于获取的数据手动创建报告，或者自动生成报告。自动生成的报告可以来自任何数据源，比如日志文件。重要的是要确保这些报告是一致的，这样阅读它们的人才会将其作为一致的信息源。接下来的几小节描述了如何创建对公司确实有用的报告，以方便跟踪流程中的安全等问题。

15.1.1 避免无用的报告

公司，甚至是个人，会在其一生中生成大量的报告，而其中只有一小部分是确实有用的。报告是一种工具，所以你必须有用它来执行有用任务的需求。有些人将报告视为提供信息的资料，但事实并不是这样。如果你不能依据报告的内容采取行动，那么这个信息就是无用的，而你可能也不会花时间去看它。事实上，无用的信息是信息泛滥现象的一个重要因素。

包含有用信息的报告能帮你执行特定的任务。它们提供了行动的基础。为此，能帮你增强应用程序安全性的报告通常包含以下这些主题。

- bug
- 速度
- 可靠性
- 使用
- 数据访问

报告的类型决定你该如何根据其包含的信息采取行动。当你不会根据一份报告采取任何行动的时候，这个报告就是有问题的，需要修改，或者是你确实不需要它。报告中的信息通常会因为以下这些原因触发某些行为。

- 模式的改变

  数据由模式组成。你可以看到数据展示方式的变化。事实上，整个数学分支都在致力于研究数据模式。当你看到数据模式上的一个改变，即使这个改变没有展现问题，你也需要调查它的源头。有些模式代表着可靠性问题，而其他一些代表着安全性问题。当然，模式的改变可能只是源于用法的改变或其他因素，但你需要知道这个模式为什么发生了改变。

- 状态的改变

  状态的改变可能意味着许多事情。但是，状态提示器会告诉你当前的信息流已经发生了改变，这意味着你需要去检查潜在的界面、可靠性、安全性或资源问题。界面问题可以导致安全问题，而资源问题往往会导致可靠性问题。

- 异常事件

  大自然不断地展现出提供异常事件的潜力。例如，雷电造成的故障是一种异常信息事件。但是，黑客试探网站以寻找网站弱点的时候，也会发生异常事件。有些黑客可以将异常事件控制在三个以内以隐藏他们自己，所以跟踪每一个事件的源头是值得的。

- 意料之外的值

  如果报告包含的信息超出了预想的范围，你需要考虑为什么应用程序会生成这些信息。意料之外的值可能来自一个 bug 或者是黑客刺探的结果。它可能是服务器上的资源达到上限或数据库中的坏数据导致的结果。事实上，意料之外的值经常出现，而人们只是忽略它们而不是采取行动。当可能的时候，要验证意料之外的值产生的原因。

去除你无法对其采取行动的信息是保持安全应用程序的重要部分。当你清楚了问题之后，就可以开始聚焦你真正需要的用于定位和验证组织安全问题的信息。关键是在创建新的报告之前，想一想你计划如何按照报告包含的信息采取行动。此外，抛弃不再有用的旧报告对于任何公司最终控制信息流都是很重要的。

15.1.2　安排时间为升级和更新做出报告

从安全和可靠性的角度来说，你需要良好的信息来针对无效的应用程序变更采取行动。为了获得你需要的信息，要将报告时间安排在你对应用程序进行升级和更新的时候。制定报告时间就是修改系统输出报告的时间或修改报告频率以更频繁地获得信息。

信息流的增加应该与你所做的变更的重要程度成正比。然而，如 15.1.1 节中提到的，你也需要避免信息过载。当超负荷时，你往往会错过数据展示出来的模式及关键事件。平衡你真正能够使用的信息和你真正需要的信息是很重要的。如果你开始发现报告出来了好几天而你没有采取任何行动，就说明你正在遭遇信息过载，并且需要以某种方式来减少信息流。

> 许多公司不够重视过滤软件。如果你能过滤个人收到的信息，使其更有针对性，就可以降低每个人承受的负载，同时也更可能以更快的速度定位应用程序升级或更新中的可靠性及安全问题。这里的问题在于要调试过滤软件使其确实能集中注意力，而不会丢失关键的信息。为了调试过滤器，要有人去比较原始数据与过滤后的数据以确保过滤器能有效工作。一旦过滤器调试完毕，就可以更多地依靠它来减少个人的工作负担，使他们更加高效。

由于升级和更新导致的信息流增加是暂时的。你只需要在合理的时间内接收额外的信息。当然，问题在于要确定何时减少信息流。通常，当 bug 及怪异事件出现的频率降低，且应用程序的使用模式开始回归正常时，你可以开始减少信息流。采用这个方法意味着你可以将一些注意力放到其他的事情上，比如计划下一次的更新或升级。

15.1.3 使用自动生成的报告

监控软件和其他支持工具通常会提供一些报告。软件供应商通常提供这些报告来响应用户的请求，所以这些报告通常能精确反映你需要用来执行普通的应用监控的数据。检查自动生成的报告清单并选择满足你的需求的报告是很重要的。选择所有的报告并希望能找到所需的信息是一个坏主意，因为这会导致信息过载并错失修正应用程序问题的机会。

这里的关键是自动生成的报告会提供一般性的信息，即供应商的所有客户都能用于监控应用程序的信息。供应商无法知道你运行的是什么类型的应用程序、用户会在何时与它们交互、用户依赖什么设备或者任何会影响应用程序执行的环境因素。简言之，自动生成的报告可以作为一个开端和催化剂，促使你对为保护应用程序及它们管理的数据所需的信息形成自己的想法。

生成的报告会提供一般性的信息，但有时候通过选择报告选项（如果有）可使其更加切合你的需求。在许多情况下，供应商会提供定制报告的方法，这样它就能提供你确实需要的信息，而不是大量你不需要的信息。重要的是要用所有就绪的功能运行一两次报告，这样你才能看到会生成哪些报告并确定你要如何使用它们。一旦你知道了这些信息，就可以开始定制报告以使其更小，用起来更高效。

15.1.4 使用定制的报告

供应商提供的关于产品的一般性报告是开始监控应用程序有无问题的好方式，但是针对你的特定需求，它们通常没有提供足够的信息。当然，供应商对你的特定需求一无所知。要获取与应用程序在你的环境中以及在用户使用中如何工作直接相关的信息，唯一的方式就是创建定制的报告。当你使用的监控软件没有提供你所需要的定制报告功能时，那就需要开发一款能够获取这些信息且能以某种有效方式格式化它的软件。这一节讨论的就是完全手动创建的报告（参见 15.1.3 节中对自定义报告的讨论）。

当然，有些公司疯狂地生成报告，有许多没有人真正明白其用途的报告。在定制报告时作出限制是很重要的。否则，你会发现自己会被埋葬在你不想要、不需要、不理解的信息之中，并且还得维护报告生成软件。下面描述了一些方法，可用来使你的定制化报告更好地反映你的需求，并只生成保持应用程序安全所需的信息。

1. 手动创建报告

在某些情况下，你要手动创建报告，从应用程序生成的日志中以及你拥有的任何其他自定义数据源中提取数据。数据源的差异很大。例如，你可能需要部分依靠用户反馈表单来创建所需的报告。仔细规划报告是很重要的，因为定制化的报告是要花时间去创建且要花很大成本去维护的。当报告依赖第三方数据且第三方有可能更改数据（产品的一部分）时，它们也可能是很脆弱的。

获取创建报告所需的数据有一些严重的缺点，特别是当处理应用程序时。数据没有免费获取的，你总得付出点什么。以下是从任何来源获取定制化数据时都应该考虑的问题。

- 输出日志数据会给应用程序或产品添加代码，这意味着要牺牲速度来换取信息。

- 黑客可以利用日志数据来更好地确定你的应用程序是如何运行的，所以你可能会向你想要阻止的人提供更加深入地入侵系统的钥匙。
- 创建日志或其他形式的应用程序或产品信息要使用你可能要用于应用程序的资源。因此，应用程序可能会遇到可靠性问题。
- 创建数据的代码可能会打开原本不存在的安全漏洞，让黑客得以入侵你的系统。
- 比起准确生成你所需要的数据，创建误导性或不正确的数据会容易得多，所以你需要时间来测试输出的数据，检查其准确性。

定制报告要服务于某个目标。但是，在创建它们时你必须要小心谨慎。只创建你需要的报告，使其只输出你需要用于采取行动的数据，并且只有在绝对必要的时候才激活报告。遵循这一策略有助于保证你的网络更加安全并减少创建定制报告的坏处。

2. 从现成的数据源中生成报告

有时候你可以从现成的数据源中创建定制化报告。有些监控产品提供了 API，让你访问他们创建的数据。如果不管你用不用，监控产品都会创建数据，那使用监控软件（或其他应用程序）的数据而不是自己创建数据是值得的。尽管如此，你仍然必须开发自定义代码，确保其以你想要的方式输出所需的数据，然后要在后续维护这份代码。当基于第三方提供的数据来创建定制化报告时，还有其他一些问题需要考虑，如下所示。

- 第三方可能会更改输出，所以你的报告也要做相应的更改。
- 数据可能会变得不可用。
- 比较难验证数据是否准确，所以你无法确定报告是否完全正确——实际上在某些情况下它可能会提供不恰当的数据。
- 为了通过 API 访问数据，你可能必须跳过许可或其他专利数据限制。
- 数据可能不是你想要的格式，所以在你使用之前，自定义代码需要执行数据操作以清理并格式化数据。
- 第三方管理数据的方式中的已知安全问题可能会让黑客得以入侵你的系统。

15.1.5 创建一致的报告

有一个问题很难解决，但很重要，需要周密考虑，那就是一致性问题。当报告以不同的方式格式化数据并以不同的顺序展示时，会造成混乱。任何使用这个报告的人在阅读它时肯定会犯错，并且可能会错误解读数据。你所使用和创建的报告应该考虑以下一致性问题。

- 数据以相同的格式出现。
- 数据的排序要尽可能保持相同。
- 报告不要使用多个冲突的方法来引用数据。
- 任何数据清理及过滤要基于一致的方式进行，为每份报告生成相同的数据（基于使用相同的算法）。
- 图形化展示的数据要对应文本数据。

当你更新报告时，请确保要对用户进行一致性问题的调查。依靠这些信息来执行任务的人会比只是管理、维护或输出报告的人更快察觉到差异。有些数据可能有很细微的差别，难以在外观和布局上发现潜在的问题。

15.1.6 使用报告执行特定的应用任务

本章已经多次谈到，报告只在其提供可操作的输出时才是有用的。但是，有些报告可能不会考虑人类读者。创建增强应用程序自动化的报告是有可能的。监控及管理程序可以读取日志文件，使用其中的数据来确定何时执行应用程序管理任务。你不想让维护行为一直发生或在特定的时间发生，更重要的是只在需要时进行维护，以减少资源的使用并提升应用程序的可用性。因为其他软件会自动读取报告里的信息并采取行动，所以你需要在创建报告的时候关注一致性和准确性。应用程序不在乎报告是否好看，它在意的是报告中的数据是否反映了应用程序的真实状态。



自动化是现代 IT 中相当重要的一部分，因为它有助于降低成本并确保行动可靠地发生。但是，黑客也很喜欢自动化，因为人们总是忘记它的存在，而黑客常常可以利用其作为入侵系统的一种手段。即使你使用的自动化是完全准确和可靠的，你仍然需要监控它来确保其能保持安全并继续按预期正常工作。

15.2 创建内部报告

内部报告是你在自己的组织内创建和使用的报告。尽管它们倾向于关注应用程序整体，但也可以让它们聚焦于你需要跟踪的任何数据源。内部报告甚至可以关注你对第三方库、API 和微服务的使用。这些报告应该能帮你执行某些有用的任务，比如跟踪潜在的安全漏洞。所有内部报告的关键是要提供应用程序变更的数据或确定应用程序是否不需要任何变更。接下来的内容会帮你更好地理解和使用内部报告。

15.2.1 确定使用哪些数据源

内部报告倾向于处理与公司相关的数据。这些报告反映了在你的系统和你提供的环境中运行的应用程序的具体特性。你可以看到用户的使用模式以及与你的系统相关的异常事件。换言之，内部报告倾向于聚焦供应商的一般性报告里无法提供的数据。

内部报告的问题在于每个人都认为其他人是值得信赖的，并且报告中的数据不会被公司以外的人知道。要认识到，如今许多数据泄露都是因为受信任的职员将便携电脑遗忘在了飞机场，或者是某些没有道德良知的人将其卖给了出高价的人。为此，你需要考虑到你在报告里提供的数据最终会暴露给外界，无论你是否希望如此。以下列出了一些在用内部数据源生成报告时需要考虑的问题。

- 确保使用报告的人确实需要你所包含的数据。
- 为使用敏感数据定义带处罚规定的协议。
- 针对涉及敏感数据的数据泄露制订处理计划（在一般的数据泄露协议之外打算执行的程序）。
- 跟踪敏感数据的使用，以便更容易确定数据泄露发生的源头。
- 保证敏感数据的安全，使其与不敏感的数据相互隔离。

> 敏感数据有时候会因为一系列相关联的错误而最终公开。保持敏感数据分离的原因就是使得它更不容易公开，比如出现在季度报告或公开演讲中。意外公开敏感数据不仅令人尴尬，还会导致各种公众问题，更不要说会给黑客入侵系统的机会。

保护数据在如今是相当重要的。有太多的公司因为犯了一个看似很小的错误最终让黑客窃取了敏感信息，结果上了行业杂志的头版或主流新闻媒体的头条。在许多情况下，错误就在于不正确地使用了内部数据源，公司本应该小心翼翼地将它保护起来，仅供不需要的人使用。在创建新的报告时你能问自己的最好问题是，能看到这份报告的人是否确实需要看到你提供的数据。

15.2.2 指定报告的使用

内部报告可以并且确实会使用敏感数据。如果你锁定每个可能敏感的信息，那么公司可能永远无法正常运转。公司的敏感信息应该在某个时间和地点公开。但是，你仍然要明确定义报告的使用方式并阐明用户应如何与其交互，以减少与敏感数据相关联的安全风险。对于处理报告中的敏感数据，公司需要有一套可强制执行的规则。

精细地考虑报告要如何使用数据是很重要的。当敏感数据在错误的场合被公开时，外推方法和数据科学方法可以让它们更加有破坏性。你同样可以使用数据清理和过滤技术来让数据变得不敏感，可以在针对整个公司的展示中使用。例如，与个人相关的数据比不相关的数据更敏感。利用清理技术可以保留数据的统计值而不会与特定的个人相关联。

报告还应该有相关的文档。你需要知道谁创建了报告以及创建原因。报告应该有一个检查日期以确保它在公司内依然有用。作为文档的一部分，你需要定义以下报告安全性元素。

- 详述谁能访问和阅读报告。
- 指定谁能生成报告。
- 定义用户要如何以及在什么地方（例如，有些报告可能不适合下载到移动设备上）阅读报告。
- 制定一个流程，确保报告的所有副本在某个时间点都被销毁（除了在安全位置保存的归档副本）。
- 提供联系信息，这样任何有疑问的人都知道要问谁关于报告及其内容的问题。

> 谈到报告包含敏感数据，BYOD（自带设备）现象给许多公司带来了安全问题。你很难阻止用户在没有重要安全措施的移动设备上查看敏感信息。你需要假设有些用户会将报告带出办公楼，无论有没有你的允许。因此，你需要准备好应对当用户丢失手机或其他包含了敏感数据的个人设备时可能会发生的泄露。给不愿遵守规则的用户设置障碍会保证部分用户诚实，但不是所有用户。

15.3 依靠外部生成的报告

跟你打交道的第三方可能需要向你提供一些报告，然后你才能确定你们之间的关系对你的公司、用户、数据和应用程序有何益处。没有这些报告，你只能猜测第三方如何看待你的组织以及你的组织如何使用第三方提供的服务。外部生成的报告的意义在于确保你明白你获得的投资回报以及第三方是否按预期执行任务。

15.3.1 从第三方获取完整的报告

你获得的任何第三方支持产品，比如应用程序或安全性测试，都需要相关的报告。报告必须详述提供的服务以及时间。它应该告诉查看报告的人为什么第三方要执行这个服务以及执行该服务的结果（例如，测试应该详述测试何时成功或失败并提供查看每种结果的方法）。你需要的任何报告都需要出现在交付清单中，第三方必须在收取报酬之前完成它们，否则你很可能不会收到你需要的所有报告。

与内部创建的报告一样，你从第三方获得的任何报告都应该促进一些行动的发生。第三方机构倾向于生成令人印象深刻的报告，而不是提供信息的报告。在处理可能由于第三方的错误或不关注细节而引起的安全漏洞时，炫目的纸张和动人的词语并不会有太大帮助。简言之，你需要像查看内部报告那样查看外部报告——总是要质疑这些报告为你和你的公司提供的价值。

可能的情况下，与第三方一起生成专门解决你的公司需求的报告，而不是接受他们为所有人创建的一般性报告。报告应该反映你的应用环境、资源的使用、人员和其他令你的公司独一无二的因素。定制化的报告可能会花费更多成本，但你能获取更多信息，而且节省的员工时间通常会带来超过额外支出的回报。

> 要尽可能多地对报告的数据进行验证测试。在许多情况下，你没有资源去检查报告里包含的每一个事实，但你通常可以进行抽查以确保第三方没有生成一份包含着他们认为你想听到的信息的报告。为了创建一个真正安全的环境，你需要可验证的事实来帮你更好地理解应用程序的动态并发现牵制黑客的方法。

15.3.2 从原始数据创建报告

有些第三方会提供 API，你可以用来访问与你的应用程序相关的原始数据。你可以使用原始数据来生成你自己的报告，而不是完全相信第三方的报告。这种服务通常会花费更多，并且不是所有的第三方都会提供这种服务，但找出这种服务是值得的。当你创建定制化的报告时，请使用 15.1.4 节中的指导原则。为了执行必需的任务，重要的是要确保你能够以你需要的形式得到所需信息。

15.3.3 保持内部数据安全

第三方可能能够访问你公司的某些内部数据。这些可能不是敏感数据，但你仍然要保护，因为你知道第三方可以访问它。当与第三方打交道时，你要将他们能访问的所有数据看作

敏感数据，遵循 15.2 节中提到的规则。除了遵循这些规则，你还要尽可能多地了解第三方。只有在执行了一些步骤，证实你确实可以信任第三方之后，你才能信任他们。以下是一些建议，用于在与第三方打交道时确保你的应用程序、应用程序数据以及公司数据保持安全。

- 验证会使用数据的所有第三方的身份。
- 进行背景调查以确保第三方是值得信赖的。
- 跟踪第三方对共享数据的访问并确保在出现异常情况时立刻跟进。
- 确保第三方签订了保密协议。
- 监控第三方以确保不会有任何试图访问敏感数据的情况。
- 制定一个处理第三方数据泄露或管理与安全协议有关问题的流程。

15.4 提供用户反馈

用户反馈在任何报告场景中都是重要的元素，因为如果没有用户，应用程序就没有用。如果你无法保证用户会对应用程序满意，那就无法保证应用程序本身是有用的。遗憾的是，少部分聒噪的用户通常会提供用户反馈，而大多数用户则沉默不语。为此，你必须使用不同的技术来收集用户反馈并确保其真正有用。一旦你做了这个决定，就可以开始使用反馈来对应用程序作出变更，从各个方面提升安全性。例如，改变 UI 来让应用程序更易于使用和理解，从而减少潜在的由于用户错误而导致的安全问题。

15.4.1 获取用户反馈

用户反馈是一种报告，并且是为应用程序维持一个安全环境的关键部分。一方面，对应用程序不满意并认为请求得不到响应的用户会自己寻找解决方案，而这些解决方案通常会导致安全漏洞和可靠性问题。另一方面，用户反馈会告诉你应用程序实际上是如何服务用户需求的。在许多情况下，用户会需要他们无法用言语表达的帮助和服务，部分原因是他们不知道应用程序没有按你原来设计的方式工作。

你可以用各种方式来获取用户反馈。在获取特定类型的反馈时，某些方法会比其他方法更好。例如，评价表单会告诉你用户对应用程序的感受，但它不会告诉你用户实际上是如何与应用程序交互的，因为用户的反馈是带有偏见的（有关这个问题的更多细节，请参阅 15.4.2 节）。当面对一组用户时，你通常必须基于收到的各种反馈进行折中。每个用户可能都感觉自己的反馈是不带偏见的，但每个人在表达观点时都会表现出偏见。这种偏见会影响你收到的反馈，进而影响其可用性。以下列出了一些获取用户反馈的方法，但重要的是要记住其中一些是具有攻击性的，而另一些可能会产生带偏见的结果。

- 创建反馈表单，让用户可以向你请求新特性或只是评论现有特性。
- 发起集体讨论，让大量用户可以谈论应用程序的优缺点并定义他们希望看到的改变。
- 给软件添加检查，记录用户与应用程序的交互行为。
- 监控用户与应用平台的交互，这样你就能更好地理解用户在应用环境中的情况。
- 跟踪用户在应用环境外部的访问，比如访问数据库、API 或微服务。
- 每当有特殊事件（比如由 bug 导致的崩溃）发生时，请求用户提供反馈。

大部分用户会对键盘记录等监控技术比较反感，因为这种感觉就像是被人监视。当使用秘密的方法来获取用户数据时，你必须考虑如何权衡给用户带来的负面感受与获取数据的好处。重要的是不仅要考虑当前应用程序，还要考虑公司的所有应用程序，因为用户会认为你一直在监控他们（即使你没有）。使用键盘记录器等产品而导致的隐私及信任丢失是你在使用这种产品之前需要仔细考虑的问题之一。也要考虑到黑客通常会利用你使用的监控系统来入侵网络。

15.4.2 确定用户反馈的可用性

获取用户反馈是有帮助的，但只有当你考虑获取数据的情形时才会有用。直接与用户交谈通常会产生带偏见的结果。监控用户会产生不带偏见的结果，但如果用户知道有监控，他的行为就不能反映真实的使用情况，你接收到的数据也将是不可信的。通过软件检查来记录用户的交互不能提供监控所能提供的信息深度。关键是你必须考虑接收信息的方式，然后再判断它是否有助于完成某个任务。

想象一下，一名用户可能会要求添加某个功能，因为它看起来很有趣。但是，测试显示这个功能可能会由于其访问数据的方式而增加数据泄露的风险。此外，监控显示用户没有因为该功能而得到多少好处——在现实世界中很少用到它。因此，你需要将三个信息关联起来以确定是否要保留该项功能。移除该功能几乎肯定会引来用户的评论，所以你需要有很好的理由移除它。在这种情况下，你需要考虑其他有用的用户反馈是否会有帮助。例如，你可能会在某个小组会议中抛出该功能的问题，然后依靠这个小组的反馈来帮你理解对这个功能的需求。

应用程序的安全性往往取决于用户的参与。社会工程学攻击是灾难性的，但试图阻止一个心怀不满的员工的破坏行为会更加困难。用户反馈的一个目标就是要确保用户对公司保持忠诚，以保证应用程序及其数据的安全。在某些情况下，为了保持用户对应用程序的好感，你需要从一开始就衡量接收可能不会有太大帮助的用户反馈的效果。

有些开发人员将小组会议视为最好能避免的一个痛苦的过程。但是，小组会议及其结论有助于带来承担安全问题的社会压力。在某些情况下，社会压力足以防止用户犯一些在没有压力时会犯的错误。当要通过某个艰难的决定的时候，小组会议还能防止你看起来像个坏人，比如当要砍掉某个可能会带来安全风险又不会给用户带来价值的特性的时候。如果使用正确，小组会议可以提供非常宝贵的用户反馈，这也会有助于建立更加安全的应用环境。

在评估用户反馈的过程中，你还必须考虑公司的政策。用户反馈通常是有偏见的，但偏见有多大通常取决于用户对应用程序的感知价值以及决策过程背后的政策。某些情况下，你需要在一定程度上忽视直接的用户反馈并尝试通过间接的方法（比如监控）获得反馈，以确定何时政策不再发挥作用，只有偏见存在。

请确保与开发团队一起考虑所有的用户反馈并让管理层知道你的处理过程。但是，重要的

是要记住你的开发团队和管理层都是由有自己的偏好和政策导向的人组成的。忽视不想听到的用户反馈，支持与自己一致的开发团队的意见是人的天性。然而，有时候用户确实是正确的，而你需要在做出决定之前考虑这一点。

在决策过程中，还需要考虑可以利用的其他信息源。你生成的报告不仅会告诉你关于应用程序、平台、第三方服务等方面的信息，还会告诉你关于公司状态的信息，并帮你更好地对收到的用户反馈作出决策。不要忘记考虑来自外部源的反馈，比如专业杂志的文章。关于某种类型的攻击正在发生的报告可能会对包含特定应用特性的做法产生较大影响，这种特性会使黑客更容易入侵系统。总之，要使用你的所有资源来获得全面的反馈，而不是目光短浅地只考虑你收到的用户反馈。

第五部分

查找安全资源

本书提供了大量关于安全威胁、来源、修复和监控技术的信息。你已经知道了黑客常用的各种方法，他们会入侵公司系统，给所有人造成麻烦。黑客造成的数据泄露形式多样，并且产生的影响也各不相同。但是，你确实需要了解关于这些问题的更多信息，而这正是本部分所要讲述的内容。

第 16 章将讨论可用于跟踪安全威胁的方法。这些威胁在不断变化，而你需要随时跟进。但是，你不想陷入信息泥潭，所以尽可能快地获取正确的信息很关键。这一章会帮你以省时省力的方式找到所需信息。

第 17 章将讨论培训的相关话题。每个人都需要培训。事实上，如果你研究大量关于安全漏洞的详细报告，很快会发现这些漏洞本可以在一开始通过更多的培训避免。当然，你需要知道安全培训的内容和对象。这一章会告诉你所有这些细节。

第 16 章

跟踪当前的安全威胁

知识运用得当，将成为相当强大的工具。为了跟进可能影响应用程序的安全问题，需要跟踪当前的安全威胁。但是，如果你关注过新闻媒体，会知道尝试跟踪所有潜在的安全威胁很耗人精力。如果你还考虑行业杂志里的报道，那么真的需要全职来做安全工作了，并且可能需要一个团队来专门管理相关信息流。很明显，应该找到识别哪些威胁确实需要关注的方法，否则会在跟踪信息上耗费过多时间，从而无法真正避免这些威胁。

一旦确定某种安全威胁确实存在，你就需要决定该采取什么行动。第 15 章曾提到，你唯一真正需要的报告是能让你采取某种行动的报告。这同样适用于安全威胁报告。需要你采取某种行动的安全威胁才是你真正需要进行彻底调查的。其他的一切都是浪费时间的多余信息。当然，在必要的时候对应用程序你得选择是更新还是升级。重点是要考虑如何应对安全威胁并在必要的时候采取行动（或只是将报告扔进废纸篓里）。

本章不是只研究安全威胁，而是探讨如何根据安全威胁信息采取行动。为了保证应用程序及其相关数据的安全，首先需要知道安全威胁，然后根据你接收到的信息采取行动。最糟糕的情况是在毫无准备、没有计划、缺乏处理威胁的相关信息的情况下遇到危机。

避免炒作、歇斯底里和无助

安全行业存在一个问题。事实上，存在许多问题。所有问题都归因为：炒作、歇斯底里和无助。安全行业因这三个词而蓬勃发展，许多安全专家也承认这一点。媒体也起到了推波助澜的作用，因为轰动的标题更吸引人。但是那些人本应让你平静、冷静、镇静地处理事情。在安全行业很容易变得失望无助，每个人都会让你一直偏执地怀疑所有事情。

炒作会持续不减，因为有了炒作才能销售虚假承诺，这些承诺有些是与产品相关的，而有些不是。例如，你可能无数次听到过某种病毒或攻击会终结互联网时代的消息，但互联网仍然存在。事实是某种安全漏洞会造成破坏并摧毁你的生活（甚至是你的事业），但是那种惊天动地的预言很少会成真。当你听到某种似乎荒谬的说法时，那它可能就是荒谬的。

即使某个说法是真的，那些在安全领域处于领导地位的人常常会提出歇斯底里的观点。你看到的轰动性标题可能会使你产生阅读内容的欲望，但它并没有真正满足你的需求。歇斯底里从来没有赢过一场战争，在安全行业也不会太有效。你真正需要做的是考虑某个威胁会对你的公司产生什么影响，可以采取什么措施消除影响，然后监控系统以确定何时（或是否）有人试图入侵。头脑冷静才会无往不胜。

这一切都会归结于一个问题，就是你会感到无助。毕竟，除非你真的感到无助，否则各种安全供应商不会承担起援救你的责任。如果本书什么也没教会你，那你起码应该知道知识才可以让你用各种方式掌控安全，而工具虽然有用，也只是众多解决方案中的一小部分。实际上你并不会无助，因为你有各种可以自由支配的资源。

16.1 发现安全威胁信息的来源

因为可用的安全信息太多，所以你需要从中挑选出能够以合适的形式提供材料、足够简洁并且不需要浪费太多时间的。安全专家的建议可以听，因为专家有充分的时间和资源去搜索这些信息。但是，这些信息需要来自跟你有类似观点的专家所管理的信息源，并且这些信息源不属于炒作、歇斯底里和没有帮助的类别。接下来会回顾最常使用的安全威胁信息源，并帮你理解每个信息源在帮助你完善安全方面扮演的角色。

16.1.1 阅读与安全相关的专业文章

你没有足够的时间去搜索所有潜在的安全威胁来源。当然，你可以让自己的团队来帮忙，另外那些在专业杂志上写文章的安全专家也可求助。这些专家花大量时间研究安全问题，他们知道各种各样的安全威胁。例如，Jeremy Kirk 最近写了一篇文章“Even encrypted medical record databases leak information”（http://www.computerworld.com/article/2980593/data-privacy/even-encrypted-medical-record-databases-leak-information.html）。这类文章需要关注，因为大部分的业务都会依赖数据库来存储信息，而存储敏感信息的数据库通常会依靠某种加密技术。我们要去查找的文章类型通常具有以下这些特征。

- 没有煽动性文字的简洁标题（尽可能做到）。
- 一段关于问题的简单概述，这样你就能确定该威胁是否需要进一步研究。
- 对黑客所采取的攻击方法的简单描述。
- 跳转到更多信息的链接，这样你不会花太多时间去查找细节。
- 对威胁所造成影响的统计（使你可以更明智地评估其对公司造成的风险）。
- 能够表明对现实目标发起攻击的可能性的案例分析。

扫描文章会花一些时间，但通过查找这些特征，通常可以避免阅读没有任何实际帮助的文章。重要的是你能在不浪费大量时间的情况下获得所需要的信息。过一段时间之后，你就能开始意识到哪些专业杂志的作者最适合你的业务需求，进而集中关注他们所写的文章，这会节省很多时间。（类似地，你也可以找出哪些作者写的文章毫无价值，可以忽视。）

> 当查看专业杂志的文章时，要记住作者的主要工作是要吸引读者以推销广告里的产品。这就是在线杂志赚钱的方式。因此，当你看到诸如“Is your connected car spying on you?”（http://www.bbc.com/news/business-29566764）这类标题时要格外小心。文章里面的内容从技术上讲是正确的，但作者会运用语言技巧来使这个威胁看起来比实际上严重得多；另一个这样的案例［文章“9 Baby Monitors Wide Open to Hacks That Expose Users' Most Private Moments”（http://arstechnica.com/security/2015/09/9-baby-monitors-wide-open-to-hacks-that-expose-users-most-private-moments/）］与婴儿监控设备有关。它们的目的是赚取阅读量，但效果则是产生了歇斯底里的现象。当然，你可能想要知道什么汽车或婴儿监控设备与 Web 应用程序有关。事实上，现在有许多 Web 应用程序运行在汽车和婴儿监控设备中，执行提供娱乐、监控环境等各种任务。即使你的公司不涉及开发这类 Web 应用程序，也适用同样的原则。

使用专业杂志的文章来定位安全威胁信息的最佳理由是，你可以在很短的时间内获得大量安全威胁的信息。只需要扫描所知道的作者写的标题类型，就能在短短几分钟内定位最重要的威胁。

使用专业杂志文章的主要缺点是，文章缺乏深度并且所关注文章的信息量较少。你要确保你严重依赖的作者是能够提供详细链接、让你无需花太多时间就能找到所有细节的人。如果你发现专业杂志的文章开始变得很有深度，那你有理由怀疑它可能想要向你推销产品而不是提供信息。尽量查找短的、有足够信息、带有大量链接的文章。

16.1.2 查看安全网站

安全网站会提供各种安全威胁的详细信息。安全网站可能不会讨论每一种当前的威胁，但你肯定可以找到大部分当前活跃的威胁（以及可能是安全专家过去粉碎各种威胁的历史）。事实上，所有这些信息的集合使得安全网站并不是你首要访问的最好选择。你真正需要做的是创建一份要研究的威胁列表，然后访问安全网站来查找这些威胁。

不是所有的安全网站都是相同的，许多在互联网上查找安全信息的人通常没注意到这点。事实上，安全网站有以下四种不同的形式（见图 16-1 的总结），每一种都有它自己的优势和缺点。

- *基于公司的网站*
  基于公司的网站常常会提供在某个领域最浓缩的信息，但缺乏其他类型的网站能支持的

信息广度。例如，SANS Institute（http://www.sans.org/）会提供大量研究材料。它还会赞助各种培训活动。这家公司（像许多其他公司一样）是受供应商支持的，但至少在看待应该如何处理安全问题上面，你不会得到单一的供应商视角。

- 政府支持的网站
  这些网站明显是基于资助它们的政府的需求而建立的，但其中大部分会提供公正的信息，适合定位当前国际性安全威胁信息。如美国计算机应急准备小组（United States Computer Emergency Readiness Team，US-CERT，https://www.us-cert.gov/）和国家漏洞数据库（National Vulnerability Database，https://nvd.nist.gov/）等网站会提供大量有用的免费提示信息和出版物。

- 社区支持的网站
  在能访问的安全网站中，社区支持的网站最有趣的，通常会提供一些在其他地方找不到的深入见解。它们会有某种议程，而这个议程会使你接收到的信息带有偏见，但是这个偏见通常不足以影响你接收到的信息的质量。比如 Vmyths（http://vmyths.com/）这样的网站可以帮你避免伴随着安全事件披露而来的歇斯底里。

- 基于供应商的网站
  基于供应商的网站通常能提供其他来源无法比拟的信息深度，因为供应商想要给你留下深刻的印象。但是，因为供应商也在积极尝试阻挡各种威胁，所以你也可以更好地了解需要做什么来准备应对某个具体的威胁。如 Symantec Security Response（https://www.symantec.com/security_response/）等网站可以快速地确定当前在互联网上的威胁的级别以及哪些威胁最有可能给大部分人造成问题。

> 基于供应商的安全网站是要向你销售供应商的安全产品。当你在浏览这种网站时要认识到：里面的文章会带有偏好，会引导你去购买他们宣传的能解决安全威胁的产品。有时候你确实需要一款第三方的产品去解决问题，但是仔细的研究、对细节的关注、合理的系统配置以及一两次更新常常会比购买一个产品更能解决威胁问题。请确保在采取行动之前检查并验证了从这类安全网站中获得的信息。在购买另一种产品之前，考虑一下用你已经可以支配的方法来解决问题。

有些网站专门整合来自其他信息源的信息。例如，Security Focus（http://www.securityfocus.com/）、Open Source Vulnerability Database（OSVDB，http://www.osvdb.org/）和 Common Vulnerabilities and Exposures（CVE，https://cve.mitre.org/）会提供能在其他网站上找到的信息的索引。使用整合网站可以节省时间，但是也会让你遗失一些重要的安全通知，因为网站的拥有者会有与你的公司需求不一样的偏好。

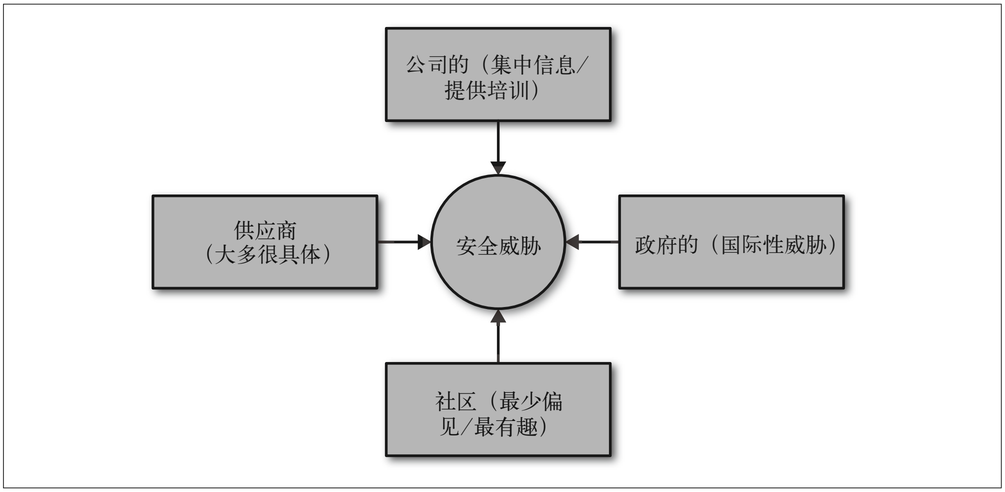


图 16-1：选择正确的安全威胁信息源是很重要的

若要研究安全威胁，只需要访问安全网站。否则，你会花费很多精力寻找可能需要处理的安全威胁的概述。请确保访问的是最能满足对某个具体威胁的需求的网站。例如，一个国家针对另一个国家发起的国际性安全威胁可以在政府支持的网站上得到最佳的报道。黑客会先加密硬盘数据然后尝试金融诈骗，这类事情通常在基于供应商的网站上有更详细的报道。如果你觉得某个威胁像是骗局，请首先访问社区支持的网站来验证它。

16.1.3　获取顾问的意见

本章到目前为止讨论的各种信息源都有一个共同之处，那就是它们提供的都是由外部信息源支持的一般性信息。但是你的公司不是一个一般性的实体，根据运行的各种应用程序、运行它们的环境以及你的用户所拥有的培训水平，会有特殊的安全性要求。在某些情况下，一般性的信息源无法提供足够的信息，它们提供不了，因为这些信息的作者对你的公司一无所知。在这种情况下，你通常需要购买某些顾问服务来处理安全威胁问题。



顾问是花钱请来的，这意味着他们提供的信息可能有更强偏见。毕竟，顾问要在生意中赚钱。有些顾问很有道德，不会故意多收钱；但顾问与你之间存在金钱交易关系，并且与其他信息源相比，顾问与你的个人关系更密切，所以需要谨慎使用顾问信息源。你需要在顾问对具体业务的更深理解与他们会提供带偏见的信息之间寻求平衡。

顾问可以提供公司所面对的特定安全威胁的特定信息。比起创建自己的策略来说，他们的建议可以更快地用来处理威胁。当然，信息的质量与价格有关。使用顾问也是处理安全威胁最昂贵的方式。对于大多数人来说，本章提供的主要信息源可以提供某些级别的免费支持。在以下这些情况中最好使用顾问。

- 你已经知道威胁很严重。

- 你无法制定策略来快速处理威胁。
- 处理这些威胁所要求的技能超出了你的公司的能力。
- 处理这些威胁需要多个团队之间的协作或需要联系其他公司。
- 自己处理安全威胁不可靠。

16.2 避免信息泛滥

黑客猖绝，所以安全问题的信息泛滥。大部分人面临着严重的信息泛滥情况。有可能收件箱里堆满了一些确实应该阅读的文章链接，但由于时间有限（比如大部分公司里都会有的无休止的会议），你无法点击每个链接。尽管如此，你还是需要花时间去指出哪些威胁是严重到必须采取行动的，这也是为什么需要从一开始就防止信息泛滥。下面总结了一些技巧，可用来避免信息泛滥并且完全避开安全威胁。

(1) 仔细选择安全信息源。确保选择的安全信息源是与公司需求相关的，能够以简单的方式展现你所需要的信息，并且能够恰当提供你需要的细节，而不会有过度的情况。
(2) 能够定位潜在的安全修复，不需要依靠应用程序的变更。在某些情况下，最好的安全信息源可以提供快速修复问题的方法，比如安装一个浏览器或其他什么的更新。
(3) 创建一个基于任务的方法来处理威胁。如果无法用一句话来概括一个安全威胁，那你就是没有理解它，也不知道自己是否需要做什么。用一句话总结威胁的习惯会让人受益终身。
(4) 丢弃喜欢用炒作手法处理安全威胁的信息源。你所需要的没别的，只有事实。当信息源开始传播炒作时，它就不在可靠了。
(5) 培养批判性思维的能力。有些聪明人会基于不采取行动时可能出现的最糟糕情况来规划他们的行动。考虑最糟糕场景中的威胁有助于识别出哪些威胁存在最高的风险，所以值得这么做。
(6) 避免昙花一现的威胁。有些威胁只在标题里出现了一次，然后你就再也没见过它了。即使对这个威胁的预测是很可怕的，但当它们无法持续太长时间，没有第二次出现在标题上时，你必须怀疑它到底会有多可怕。

为避免不堪重负，请确保花必要的时间去理解威胁实际的工作方式。例如，你可能听过像 rowhammer（http://www.rowhammer.com/）这样可能给系统带来灾难的威胁。但是，当你开始深入了解时，会发现这是一个有趣的攻击行为，但它需要某种类型的系统访问权限，而黑客在面对 Web 应用程序时通常没有这种权限。因此，你现在可以忽略 rowhammer 了。当然，一个很棒的安全资源会从一开始就告诉你事实是怎样的，但有时候你需要自己去研究，以避免被没用的信息所淹没。

如果可以，请一定要使用数据过滤。用关键字对收集的信息源进行快速搜索，可以帮你确定一个威胁是否真的是你应该考虑解决的威胁。现在市场上有许多可以用各种方法过滤和识别信息的工具。知道如何高效地利用这些工具可以在长期的运作中节省不少时间，而你现在只需要花一些额外的精力去学习这些工具。

16.3 为基于威胁的升级制订计划

当你以足够快的速度收到关于某个潜在威胁的信息，而修复这个问题需要改动的代码量很大时，可能需要升级应用程序。当然，你必须首先明确是否需要采取行动。有一些威胁确实要引起注意，但它们不会影响应用程序，因为需要某些特别的访问权限或者只会影响你根本没有用到的软件。接下来会帮你明确某个基于安全威胁信息的升级是否真的需要，以及确定需要之后如何升级。

16.3.1 预判不需要采取任何行动的情况

研究安全威胁信息文档时，都会觉得这个威胁听起来很可怕。但它实际上只是对某些人可怕，对于你来说可能根本就不是什么问题。有些公司常常要应对不会给它们造成问题的威胁。简言之，当不需要做任何事情时，他们却陷入到围绕安全威胁的歇斯底里之中去了。当你检查当前的安全威胁时，需要考虑媒体上提到的危险是否真的在你这里出现了。在许多情况下它们并没有出现，不需要对它们做任何事情。置之不理即可。

当然，问题在于你要能达到一定水平才能知道某个威胁会不会影响你。有错失威胁很严重，因为它会造成攻击以及之后的很多问题（比如网站崩溃或数据泄露）。评估某个具体的安全威胁会带来的潜在风险涉及以下事项。

- 确定你是否使用了被描述为引起安全威胁的技术。
- 考虑一下引起威胁所需的一些访问是否真的发生在你的公司身上。
- 明确在哪种环境下你的公司会遭受攻击以及攻击何时可能发生。
- 检查现有的防卫手段以确定它们是否足以阻挡攻击。
- 确保当攻击发生以及你错估了黑客发起攻击的能力时有应对的策略。

16.3.2 决定升级还是更新

第 13 章和第 14 章讨论了升级和更新。但是在现在我们讨论的这个情况里，你要考虑的是由于安全威胁而引发的升级或更新。有很多相同的规则可以应用。升级一般预示着某些重大的代码变动，而更新可能完全没有代码变更，但可能会反映 UI 的改变或其他一些无需修改底层代码就能快速更改的元素。因此，界定升级还是更新的首要方法跟你在进行其他变更时采用的方法是一样的。

安全威胁可以改变升级或更新的意图和紧迫性。在某些情况下，需要处理以下几个方面的问题。

(1) 以更新的形式处理安全威胁聚焦的问题。
(2) 升级或更新正在支持的软件（比如防火墙或操作系统）以确保安全威胁不能从另外的方向发起攻击。
(3) 培训用户，避免他们执行任务的方式会提供给黑客有价值的信息或访问途径（黑客可利用其发起攻击）。
(4) 修改目标代码周边的所有代码，确保应用程序在发布更新之后能按升级设计的那样正常工作。

(5) 执行所有依赖项的升级，确保整个应用程序稳定。
(6) 用黑客会使用的方法进行广泛测试以验证升级后的应用程序可以继续阻挡安全威胁。

不是处理每一个安全威胁都要执行这一大串步骤，但要在决定如何处理时考虑每一个步骤。你可能会发现只要一次更新就足以阻挡威胁，或者一次操作系统的升级就是你真正需要的。关键是要考虑安全威胁与你的应用程序及其支持的数据有关的各个可能的方面。

有时候你真的需要就使用升级还是更新来修复安全威胁作出快速的决定。这一部分所述的步骤当然是有用的，但图 16-2 里的流程图可以提供更加快速的作决定的方法。

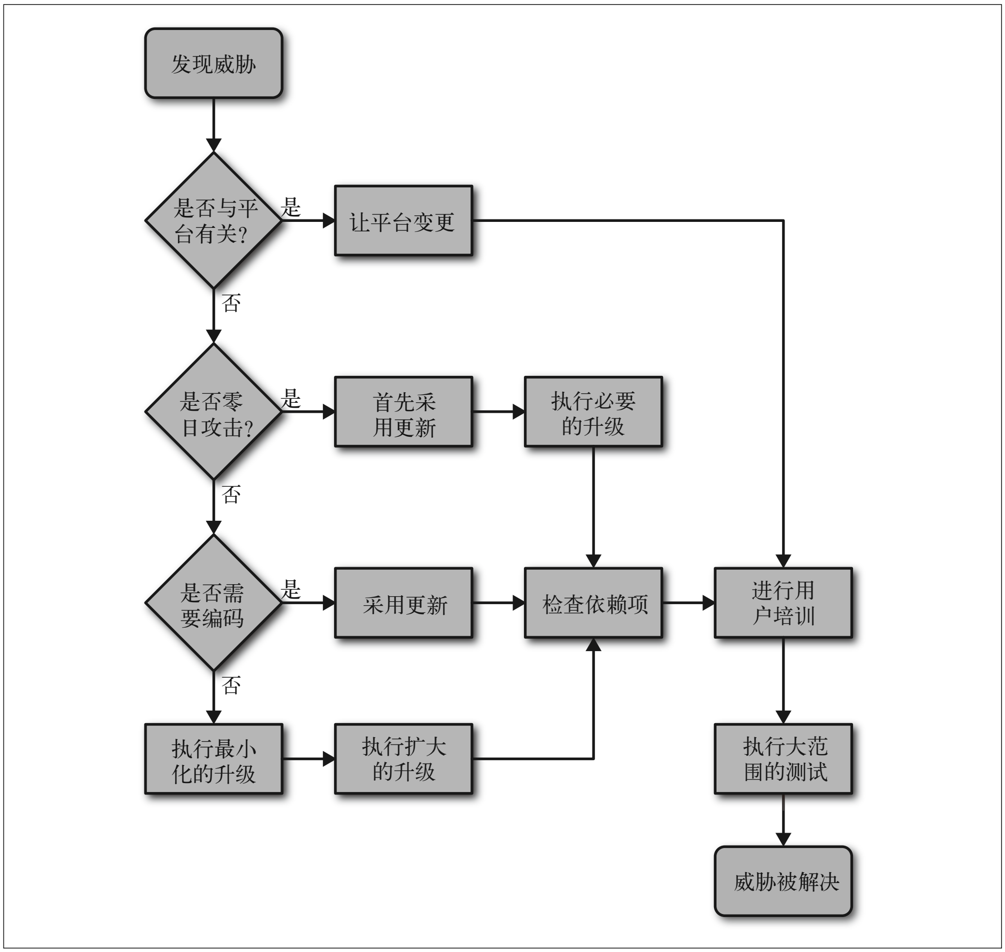


图 16-2：快速决定升级还是更新有时候是必需的

当面对零日攻击时，通常最好的方法是在尽可能短的时间内执行自己创建的更新并测试。黑客不会在攻击你的系统之前等待你发布更新。事实上，黑客会希望你陷入到流程和公司政策当中，这样他们就有更多的时间去攻击你的应用程序。速度是阻挡黑客的一个关键因素，但请确保在每次发布之前测试所有的软件以保证你不会使事情变得更糟。消除安全威胁要求尽快发布精准的策略。

16.3.3　定义升级计划

第 13 章描述了应用程序的标准升级过程。它详细描述了一个需要几个月来完成的升级过程。当然，你在进行这种应用程序的升级时，有时间去仔细和完全地执行各种任务。安全威胁很少会给你几个月的时间去创建升级并发布。能获得预先警告来执行长期升级的唯一方式是，正好有人能够在安全威胁信息一出来时就获得它（许多有道德的黑客会在将信息公之于众之前给供应商三个月到一年的时间去修复它）。

由专门进行安全测试的公司对应用程序进行持续测试可以提供给你更多升级的主导时间。安全测试公司可以在应用程序中潜在的缺陷变成公开信息之前发现它们。当然，为了得到这种提前的信息，你必须要愿意为这种价格不菲的服务付款。

当用升级来响应安全威胁时，为了在短时间内发布升级，你需要考虑创建升级和逐步测试的方法。敏捷开发技术（http://agilemethodology.org/）能帮你完成这个任务。当然，如果你的公司平时没有使用敏捷开发技术，花时间去学习它也没有太大的作用。尽管如此，升级计划还是应该包含一些能比平时更快地开发代码和测试的方法。在威胁应对过去之后，需回头检查代码，并确保没有在修复这个安全威胁带来的漏洞过程中引入更多的安全漏洞。

14.4 节讨论了一个快速响应团队的需求。在许多情况下，使用这个团队来把升级放在一起会节省时间，也可能会增加费用。这个团队已经熟悉如何快速地处理更新。使用它来进行升级以应对某些安全威胁是有意义的。但是，你要确保这个团队里有合适的人员来处理升级。你有可能需要增加团队成员来提供一般更新不要求的服务。

许多公司面对的致命问题是变得歇斯底里并要求持续更新的管理者。“我们好了吗”这样的问题绝对会拖慢升级的过程。负责升级的开发团队应该定期向管理层报告，但管理层也应该认识到需要让开发团队平静地工作。否则，当一名重要的团队成员在做一份不必要的报告时，黑客就有可能入侵你的系统造成噩梦。

16.4　为基于威胁的更新制订计划

在某些情况下，你会在很短的时间内收到安全威胁信息，需要现在就进行一次更新并在稍后进行一次升级。更新很少会改变应用程序的代码并且有时候根本不会影响代码。作为基

于威胁信息的更新计划的一部分，你需要考虑更新是否真的很紧急。此外，你还需要考虑对你在应用程序中所依赖的第三方软件的影响。

16.4.1　验证更新是否可解决威胁

本章介绍执行的速度。当你识别出一个安全威胁时，需要尽快对信息作出响应。因此，更新是解决这个问题的最佳方法，因为更新需要更少的时间、更少的资源，并且在测试期间可能造成的问题更少。在这种情况下，更新不只是意味着针对你的应用程序的更新。更新可以包括以下事项。

- 调整已有安全软件
- 更改宿主平台
- 修改宿主浏览器
- 通过用户培训或其他方法促成流程的变更
- 通过非代码或轻量代码变动的方法来对底层应用进行更新
- 修改依赖的软件（比如库、API 和微服务）

遗憾的是，不是每次更新都能修复问题。如果目标是要阻挡一个安全威胁，那么最佳的处理方式是首先找到正确的行动方向，因为浪费时间尝试其他方法只会延迟有效方法的采用。黑客不会一直在等待。有时候处理速度要慢下来以进行更多细节的修复，并且承认你需要更多的时间，而不是能够轻易修复好。

问题的核心在于要找到黑客使用的攻击类型。例如，中间人攻击可能只需要你正确地加密传入和传出的数据并加倍小心。此外，它还要求额外的认证和授权方法。这类攻击无法用更新轻易地修复，因为你要查找数据准备、发送和处理的方式，而这三个任务都是底层代码必须执行的。

DDoS 攻击可以用更新来解决，更改安全软件的设置、提供额外的系统资源以及重新配置平台，这些措施都可以使这样的攻击难以奏效。你也能够通过用户培训、修改 UI 以及更改应用程序的安全设置来修复社会工程学攻击。攻击的类型决定了更新是否合适。在许多情况下，你会发现通过更新你至少可以降低攻击发生的可能性，但可能无法完全修复它。

处理应用程序的停机时间

大部分人不愿谈论应用程序的停机时间，特别是与安全问题相关时。当应用程序停机时，人们会开始问各种问题，而问题通常会导致怀疑（真的还是假的）。在某些时候，问题变成了你是否要在有攻击的情况下继续让应用程序运行，还是你是否必须将其关闭以避免在稍后又遭受伤害。在某些情况下，黑客不会给你选择的机会；攻击本身会导致应用程序以某种特殊的方式崩溃。关键在于黑客确实会使应用程序停机，而这永远不会使赞成停机的 IT 人员愉快。

尽管广告里经常看到某些公司宣传 99.999% 的正常运行时间，但停机现象确实存在。当然，大公司的停机事件会出现在主流专业杂志文章里，让企业更难堪。有一件事很有趣，这些文章通常会包含导致停机的模糊（并且可能是不正确的）理由（请参阅

http://www.zdnet.com/article/microsoft-explains-roots-of-thisweeks-office-365-downtime/）。但是，有时候停机以及随后的数据丢失是因为其他原因，比如自然的力量（请参阅 http://www.computerworld.com/article/2974260/data-center/mother-nature-teaches-google-a-lesson.html）。关键在于，停机会由于各种原因发生在每个人身上。

当攻击使你开始担心数据丢失、系统完整性损失或者是严重的破坏时，你可以考虑让应用程序甚至整个系统停机。这样做就有时间去执行所有必要的更新，然后尝试让系统重新上线。虽然停机被管理层视为消极的处理方法，但有时候它是防止出现严重损失的最佳方法，这些损失得花上几周或几个月去弥补（通常会涉及更多的停机）。

16.4.2　确定威胁是否紧急

在谈论安全威胁时，很重要的是要确定威胁是否紧急。第 14 章讨论了从普通更新的角度来看的关于紧急的标准。除了一般的考虑，在处理安全威胁问题时还需要考虑以下事项。

- 主动以你的系统或应用程序为目标的任何零日安全威胁。
- 任何进行中的攻击。
- 直接向公司或其某个职员发起的重大、可验证的威胁。
- 针对你所依赖的库、API 或微服务的攻击。

16.4.3　定义更新计划

第 14 章定义了更新计划的要求。在处理解决安全威胁的更新时，你要使用快速响应团队来确保更新是完整的并且以及时的方式被测试（请参阅 14.4.2 节）。确保你考虑了 16.3.3 节中讨论的同样类型的问题也是很重要的。例如，使用敏捷开发技术会使事情进展得更快，并减少获得好的更新所需的时间。

16.4.4　要求来自第三方的更新

有时候许多开发团队在匆忙进行升级时会忘记处理依赖项的需求。如果依赖代码是可信赖的且你正在使用最新的版本，可能不必去立刻检查更新，但确实需要让第三方认识到它在产品中的潜在安全问题并要求解决这些问题。否则，任何修复可能都无法继续发挥作用，黑客会选择另外的攻击手段。

第17章

获取必需的培训

即使是遵循最佳实践并使用了各种最新反黑客策略的安全应用程序，也很容易因为用户没有安全意识而被攻破。社会工程学攻击仍然是入侵组织最有效的手段。这并不是说用户愚蠢或没有能力作出好的选择，事实是他们通常没有接受适当的培训并且缺乏巩固所学知识的需求。仅仅为了完成任务而进行培训不会取得任何成果，除非知识经过反复检验，期间有过失败（或者成功）。

但是，开发人员、系统管理员、管理层以及公司里的所有其他人员也有培训需求。没有适当的培训，你就不能期望公司能够阻止黑客。开发人员需要密切留意新的威胁并修复安全漏洞，系统管理员需要对潜在的攻击行为保持警惕，管理层需要知道提供资源来保证应用程序安全的必要性。总之，每个人都有需要经过一定级别的培训才能完成的任务。

本章会讨论一些帮助相关人员得到培训的方法，这样应用程序可以执行得更好，黑客也不太容易发起攻击。培训不会解决所有问题，但是将良好的培训与安全的应用程序、合适的支持软件以及设计良好的库、API 和微服务相结合，会让黑客重新考虑是否要发起攻击。坚定的黑客总是能够破坏你的安全性，但如果你能给黑客一个很好的理由去另外寻找缺乏有效防御措施的目标，这肯定是值得的。

制订专门的培训

本章使用**培训**一词泛指各类培训，因为所讨论的大部分问题适用于所有类型的培训。无论进行什么类型的培训，都必须给培训留出一些场地空间。

某些培训确实具有普适性。例如，公司的每个群体都需要接受公司安全流程方面的培训。无论这个群体是包含用户、开发人员、系统管理员还是管理层，每个人都需要知道公司的规定以确保安全流程正常运行。

但是，某些培训是非常具体的。用户可能需要有关数据输入的培训。为此，你要调整培训区域的大小，使其适合于只包含用户的小群体。开发人员在大多数情况下是一个更小的群体，他们可能需要专门针对访问库、API 或微服务的培训。可能有些培训信息是敏感的，所以你需要划定更小的培训区域以达到所需的保密程度。

针对群体的需求制订专门的培训是很重要的。确保满足这些群体的隐私、规模和技术需求也是很重要的。在你寻找具体的培训场地时，要考虑培训对象会如何学习。例如，如果该群体要求使用投影仪，那么你需要一个有适当设备和足够电力的房间来满足这一需求。本书不是关于演讲的，但是在没有首先满足群体中每个人的培训需求时尝试讨论安全性会使你的演讲以失败而告终。

17.1 制订内部的安全培训计划

根据公司规模大小，一份内部的培训计划比起聘请第三方的培训师或培训学校可能会更省钱并且效果更好。这个方法的好处是你可以讲授应用在你的公司里的安全措施，这意味着你的员工可以在很短的时间里获得相关的信息。其缺点是内部的培训师可能没有专业培训师所拥有的技能、教育背景或经验。接下来我们讨论内部安全培训的需求。

与安全工作的其他许多方面一样，选择内部安全培训时必须考虑风险因素。确实，培训师知道公司所有的安全性要求，但是公司提供的文档可能不完整或者是错误的。风险就在于会给你的员工提供错误的信息，即使这些信息与你的公司认为正确的原则保持一致。使用外部培训通常会使员工谈及最新的安全趋势，并且可能促使你的员工就安全计划作出改变，以更好地促进公司内的安全流程。

17.1.1 定义所需的培训

在为公司举行培训活动之前，你需要知道员工需要哪些培训。公司里的每个人都需要某些级别的培训，即使是那些已经接受过专业培训的人。创建一份所需培训的清单可以调整培训课程并让你的投入获得更高的价值。此外，因为公司的培训资源通常是非常有限的，所以你需要从它们身上获得最大的价值。为此，考虑将以下这些问题作为培训计划的一部分。

- 员工当前的技能水平
- 普遍的培训需求（整个行业的需求）
- 针对具体公司的培训需求（比如表单或流程的使用）
- 可参与的员工
- 可用的培训师
- 培训师对员工需求的熟悉程度
- 可用的培训地点

除非你为达到培训目标创建有用的计划，否则员工将一无所获，培训师会有挫折感，而你

投入的时间将毫无价值。重要的是要理解，内部培训的好处只是在于可以利用内部环境来量身定制满足需求的培训。例如，在一个安静的房间里，培训师可以一对一地对有特殊需求的员工提供培训，其结果会比在一个嘈杂的教室里要好。

17.1.2 设置合理的目标

内部培训的主要问题是，管理层会设置不合实际的目标，而培训师又缺乏引导管理层走向正确方向的经验。因为员工可能会不愿意参与培训，所以他们有时候会觉得培训师应该让他们休息（以参加培训的名义）。没有明确定义的目标无法使任何人获得他们想要的东西，他们甚至不知道自己获得了什么。没有合理的、明确定义的目标通常会导致精心准备的培训课程失败。总之，目标的设置是进行培训工作的关键。以下是一些在设置公司培训目标时需要考虑的事项。

- 基于公司真正的培训需求设置目标，而不是基于可有可无的需求。
- 明确每一个人都能理解的目标，使用大家都能理解的语言，而不是行话。
- 确保目标与员工的学习能力一致，而不是你认为他们应该学习什么。
- 设置目标以有效利用你的可用时间，并留出额外的时间给更难的主题，因为员工肯定会有疑问，或者不理解你的材料。
- 提供时间给需要单独培训的员工和错过了培训时间的员工。
- 考虑培训中所有相关人员的意见：管理层、培训师和员工都应该在培训课程中有一定的发言权。

只有当培训计划包含合理的目标时，才能期望获得成果。重要的是要明白，你可能无法达成所有目标，所以给目标清单设置级别（达到一定级别的目标仍然被认为是成功的）很重要。你总是可以在稍后开展另外的培训以解决难度较高的目标，而重要的是要在培训课程中获得可以展示的结果，这样当管理层问起培训情况时你可以指出完成的目标。

> 看起来似乎培训师应该为培训设置目标，但在许多情况下这并不是最好的做法。培训师需要花时间准备给员工培训新技术，所以协调培训课程会给培训师带来不必要的负担。指定另一个人来协调大家的工作，这在许多情况下效率更高。这样，所有相关人员可以关注各自要关注的具体领域。

17.1.3 使用内部培训师

许多公司要求员工进行必需的培训。在许多情况下，使用内部培训师是可以接受的，但有些情况需要聘请顾问或专业培训师以获得好的结果。使用内部培训师有利有弊。其优点如下。

- *低成本*
  在大多数情况下，使用内部培训师会比聘请专业培训师便宜得多。

- *熟悉程度高*
  内部培训师已经很熟悉公司规定以及实施这些规定的员工，所以更容易提供个性化的指导。

- 实用

  采用内部培训意味着可以根据公司的需求而不是培训师的需求来安排培训日期。此外，有疑问的员工在需要的时候可以直接找到培训师。

- 方便

  在大多数情况下，使用内部培训师会比在公司外部寻找有相关知识的培训师更方便。有效地聘请外部人员意味着必须花时间为不会成为公司员工的人安排面试。

- 知根知底

  公司已经知道了内部培训师的知识水平、可信任程度以及能力。而有些在面试过程中表现良好的外部人员会在进行实际培训时表现很差，低于你的预期。

为了利用这些优点，管理层必须使培训师和员工愿意参与到培训过程中来。即使在最理想的情况下，使用内部培训师也至少会有以下这些缺点。

- 缺乏尊敬

  员工和培训师已经非常熟悉。教室环境要求培训师获得尊敬，而其他员工可能会拒绝给予这种尊敬。

- 缺乏时间

  内部培训师会将精力投在培训任务上而不是处理日常业务上。你无法期望他在这两方面都表现颇佳。这意味着在培训课程前后你会失去一名员工。

- 缺乏技能

  通常来说，任何内部培训师在技能方面都不能与全职进行培训的人旗鼓相当。科班出身的培训师是个例外。

- 缺乏经验

  即使某人有所需的技能和教育背景，但没有全职参与培训任务这一事实意味着他缺少全职培训师所拥有的经验。

17.1.4 监控结果

培训是一个监控和调整的过程。不能简单地将信息灌输进人们的脑子并期望会产生作用（至少现在还不行）。学校要进行考试的理由就是帮助老师明白在当前环境下的不足并帮忙进行调整。培训同样如此。为了取得最佳的结果，必须监控培训工作并作出调整，以适应不同的人以及他们对主题的不同理解。当然，与工作相关的考试可以有许多形式。以下是一些监控培训有效性的做法。

- 书面考试

  学校用考试来衡量培训的效果，这在企业环境中也同样有效。当然，有些人知道如何应对考试（其实并没有真正理解培训内容），而有些人则相反（虽然非常理解培训内容）。所以，这不应该是评估培训成功与否的唯一方法。

- 实际动手的测试

  另一种检查结果的方式是创建一个测试场景并让员工展示他们接受到的培训。在书面考试中不及格的人常常在实际动手的测试中做得更好。但是，这类测试仍然有与书面考试

一样的问题：有人就是能够在不真正理解培训材料的情况下操作得更好。

- 实践因素
  培训应该在整体工作效率和生产力方面产生明显的效果。简单地查看培训如何影响业务可以证明培训是否有效。例如，明显下降的安全相关错误数表明安全培训达到了预期目标。

- 监控结果
  从整体上查看业务情况可能无法让你完全了解培训效果。有时候，你需要监控特定的业务领域（比如电子邮件攻击成功率的下降等），才能知道培训是否按预期起效。

- 交叉培训
  教授其他人正确地执行任务是许多公司没有想到的评估培训成功与否的一个方法。为了培训其他人，员工必须很好地吸收知识以有效地运用它。交叉培训可以证明员工真正理解了培训材料并且能够表达出来让其他人理解。

> 在监控过程中会出现的一个严重问题是，有人会有受威胁的感觉，并且工作效率会降低。举一个例子，当没能达到某个特定的培训目标时，似乎每个人都要去责备某人或某件事。但是，失败只是表明需要更多的培训，而不是需要责备（请参阅文章“Defining the Benefits of Failure”，网址为 http://blog.johnmuellerbooks.com/2013/04/26/defining-the-benefits-of-failure/）。失败的原因有很多，而不只是由于一个人的错误。浪费时间责备他人是不会产生生产力的。更好的做法是正视失败发生的事实并提供补救的培训。

当你监控培训的结果时，可以了解员工实际掌握了哪些知识。针对没有达到培训目标的知识，你可以重新创建目标并尝试其他讲授培训材料的方法。重复相同的培训方法通常会产生相同的结果，对相同的材料制订不同的培训方法对于确保员工真正吸收它是很重要的。

当培训正在进行时，你可以利用检查已达到的目标的机会，在员工已积累的知识之上设置新的目标。重要的是不要创建大量的目标清单，这会让学习任务看起来无法完成。你需要做的是创建一些能够很快达成的小步骤，这样员工可以保持对培训的积极态度。

培训方案的最终目标是保证应用程序及其相关数据的安全。就像本书在前面提到的，公司获得安全环境的唯一方式是获得员工的支持。通过培训，可以获得员工对应用程序的信任，并确保每个人对安全在工作环境中扮演的角色有更好的理解。获得这种结果的唯一方式是保持积极的环境。

17.2　获取第三方对开发人员的培训

第三方的培训通常能提供针对一般性安全信息的培训，这能够很好地满足大多数公司的需求。当然，比起使用内部培训师，这种一般性培训会有更多的开支，所以你在花更多钱的同时却只能得到较少与你的公司相关的信息。但是，在大多数情况下，员工接受的培训会比内部培训更专业和及时。通过第三方，你获得的是培训的质量。

不是所有的第三方培训都是相同的。获得第三方培训有多种方式，每种方式都有利有弊。下面列出了最常见的第三方培训方式。

- *内部安全培训*
  有培训经验的安全顾问来公司并在你指定的区域内提供服务。你可以根据员工需要的信息类型，将反馈提供给培训师。因此，你可以在一定程度上定制培训内容，并且在一般性信息和与公司相关的信息之间获得更好的平衡。此外，员工可以在熟悉的工作环境中使用他们平时使用的工具进行学习。这个方式的一个缺点是你必须提供培训场地，并且这个场地要大到足以容纳你希望参加培训的员工。此外，这也是最昂贵的培训方式。

- *网络学校*
  采用这种方式时，员工通常可以按他们自己的速度接受培训，这可能意味着更好的培训效果。但是，这种方式需要员工进行自主学习。其好处是它比其他大多数方案要少花很多钱，并且在大多数情况下可以提供高质量的培训。其缺点是员工不能在课堂环境中学习，可能在某些情况下会发现问题难以得到解答。你还可能发现培训材料所受的限制更多。

- *培训中心*
  培训中心提供专门的课堂环境和专业的安全培训师。因此，这个方案能够提供最好的一般性培训，员工可以与培训师有最好的交互。培训中心通常会将培训班保持在很小的规模，这样培训师可以与每个学生交流并且能够鼓励他们做到最好。这种培训的开销稍大：当培训中心的距离较远时，你可能会发现要为参加培训的员工支付额外的开支。

- *学院和大学*
  在大部分情况下，员工获得的是与他们在上学时获得的同样水平的培训。但是，课堂可能会很拥挤，老师可能会不堪重负，材料可能已过时。在某些情况下，这种方式是免费的，或者只需要支付所需的课堂材料。其缺点是学校开办的课程是最适用于学校的，而不是你的公司，所以你最终可能会失去参加培训的员工。

你可能无法找到最完美的第三方培训方式，但通常可以找到最适合的一个。你的目标是要在你能支付的范围内找到所需的培训级别（或者尽可能接近的级别）。接下来看看第三方培训师如何帮助你的员工获得所需培训。

> 某些级别的培训会提供内部培训师无法提供的证书。证书可能对你的公司没有意义，但它可以用于向潜在的客户证明你的员工接受过必需的安全培训。拥有证书让你对某些客户更有吸引力并且可以提高业绩，从而赚回花在培训上的钱。

17.2.1 指定培训的要求

在与第三方打交道时，你仍然需要执行 17.1.1 节和 17.1.2 节所描述的那些任务。但是，你现在必须为根本不了解公司文化和需求的人执行这些任务。为了获得所需的培训要求，必须提供一份蓝图给提供培训服务的人或机构。任何涉及寻找和查询第三方培训师以及与之交互的员工都必须完全理解培训的要求并清楚地传达。否则员工接受的培训不会有效，并且你会继续遇到本来可以通过适当的培训避免的安全问题。以下清单描述了在与第三方培

训师交流时应该考虑的事项。

- 讨论培训时间以确保培训师能为公司提供服务
- 指定培训场地
- 确保培训师确实能提供所需的服务
- 确认培训师有适当的资格证书
- 为了获得想要的结果，创建培训师需求清单
- 获得所需的特殊设备的清单

> 第三方培训师和培训机构能够提供的价值各不相同。请确保你获得承诺的服务要有书面记录并且要对获得的培训服务进行记录以证明其已完成。如果有可能，花时间与他们以前的客户交谈以发现潜在的问题。当然，培训师推荐的人几乎都会讲正面的事情，因为他们很可能有正面的体验。尽管如此，通过仔细与过去的客户交流，你还是能获得足够的信息以作出正确的判断。与任何其他业务一样，在聘请第三方为公司进行培训时保持谨慎是很重要的。

17.2.2 为公司聘请第三方培训师

当聘请第三方培训师进行教学时，你需要举行几次会议以确保培训师理解你的需求并确认你已为培训师做好所需准备。培训师来到现场授课的第一天不应该出现浪费大家时间的悲剧。提前做好功课会减少第一天的问题，这样每个人都会有好的开始。以下是一些在与第三方培训师讨论时要考虑的事项。

- 讨论公司存在的安全问题并展示这些问题如何在公司内发生。让培训师准确地知道在培训课程中哪些安全领域要处理和强调，这一点很重要。
- 确保培训师理解安全问题背后的历史并知道你在过去采取的措施。这一步确保培训师不会浪费时间去尝试你已经尝试过或排除过的事情。
- 说明培训师在培训过程中所受的潜在限制，确保员工在接收到与公司规定不符的信息时不会反感。
- 确保你挑选的培训地点满足培训师的要求。此外，检查培训师需要的设备类型以及培训师会提供什么设备。

与访问公司的其他客人一样，你需要确保培训师感受到欢迎，但不能自由访问敏感区域和敏感信息。培训师不能以某种方式凌驾于公司的安全要求之上。事实上，有些培训师从事该职业的目的就是发起社会工程学攻击。请持续保持对第三方培训师的尊敬，但也要保证培训师受到适当的监控，这样你才能保证公司的安全。

17.2.3 利用网络学校

网络学校很少提供定制化方案，你所看到的课程就是你能得到的课程。请确保网络学校可以提供某些异步方式（比如电子邮件）来与老师取得联系。要确保员工在方便的时候可以联系上老师。很重要的是，要认识到网络学校的交流是有限制的，所以你可能需要安排一

名博学的员工随时提供帮助。

像之前提到的，网络学校的好处是每个人都可以在合适的地点接受培训。但是拖延是人的天性，许多人会在最后的时刻临时抱佛脚。这种形式的培训需要有管理层的额外支持和博学员工的监控。你应该明确看到员工在培训过程中掌握知识的进度。否则，你需要怀疑某个员工是否完整参与了培训过程。



很重要的是要认识到，不是所有人都能进行在线学习。人们可以有各种学习方式。事实上，某些人在课堂上根本学不好，他们需要实际操作的培训或看别人如何执行任务。你可能会发现有些员工不能进行在线学习，而是要依靠内部培训师或其他方法来获得所需的培训。问题并不是某个员工没有努力尝试，而是使用某种培训方法时的能力差异。

17.2.4　依靠培训中心

只要员工愿意，培训中心可以提供最好的培训。关键是要记住，这种培训不仅昂贵，而且内容也没有针对性。培训中心通过使用脚本化的技术提供强化培训并以此赚钱。这种培训确实是很好的，但员工必须愿意完全参与到这个过程中并在其他时间（课后）继续学习。

为了使这种培训方式起效，在培训阶段不要指望从员工那里获得任何产出。中断只会使员工分心并降低培训的效率（考虑到这些地方的收费价格，中断真的是非常昂贵的）。因为培训的时间和场所是不可协商的，所以你可能需要以各种方式补偿员工以确保培训能正常进行。

当员工参加完培训回来时，请确保完整测试他们所获得的新知识。此外，让员工就新学到的技能进行演讲通常对他们是有帮助的。这个方法有助于强化员工获得的知识并给其他员工传授新技术。演讲属于 17.1.4 节所描述的交叉培训的范畴。你可以使用那一节描述的其他技巧确保从投入到培训中心的钱中获得最大价值。

17.2.5　利用本地的学院和大学

不是每个公司都有时间、资源或资金使用这一部分所描述的各种培训技巧。你的员工仍然需要培训。即使你无法提供内部培训和支付本章列出来的其他培训方案，但是仍然有一个好的选择，即支付本地学院或大学提供的服务。根据你所在地区的教育政策的不同，你可能需要为员工支付一定的学费，但会发现这比其他的方案要便宜得多。

学院和大学的教学不会有立竿见影的效果。员工会按照统一的进度接受培训，所以这个方案不能满足短期的培训需求。采用这个方案的员工可能会上整个学期或整个学年的课才能获得所需的信息。所以你需要对这个方案有先期计划。

这种培训通常比去培训中心更灵活。例如，员工可能会发现这门课程在一天内会开放几次，并且晚上也是可以的。尽管如此，你通常会发现排期并不如本章介绍的其他方案那样灵活，学校肯定会优先满足自己的需求。

与选择培训中心一样，请确保你的员工可以展示所学到的知识并进行演讲以交叉培训其他

员工。因为培训的时间很长，所以通常可以在培训期间分阶段做这些事情。此外，较长的培训时间意味着员工不用匆忙获取新的技能，而有更好的机会巩固知识，因为有更多的时间吸收培训课程里的材料。

17.3 确保用户有安全意识

每个人都需要接受一定级别的培训以确保公司的安全性。但是，用户通常需要特殊的培训，因为他们不具备开发人员在获取认证或学校学习时掌握的知识。此外，用户缺乏开发团队或其他员工在多年处理安全问题中积累的经验。因此，为了创建一个安全的环境，在进行用户培训时需要制订一些额外的计划。接下来就如何增强用户的安全意识提供一些方案。

17.3.1 制订专门的安全培训

今天我们使用的安全培训的一个问题是它不具有针对性。你必须使安全培训专门针对公司并能反映公司的规定。否则，你无法期待员工会在日常工作中遵守这些规定，而结果通常就是出现本可以通过遵守规定避免的安全漏洞。

培训还需要考虑法规或其他要求。不是每个公司都有相同的安全要求。为了让员工们都知道在公司里要如何安全地执行任务，你需要使他们了解所有会影响公司的法规和其他要求。重要的是员工不仅要理解这些内容，还要理解公司是如何满足这些要求的。

另一点也很重要，那就是聚焦公司具体的问题领域。使用真实的安全案例（包括名称、日期，而有些细节需要被移除以保护隐私）。通过聚焦公司的具体需求，你可以让培训更有针对性，并且能够确保员工看到遵守培训定义的要求所带来的好处。更重要的是，作为培训的一部分，可以帮助员工理解安全漏洞对他们自身的影响。关键是要让员工理解在涉及安全时遵守公司的指导原则对他们个人利益的影响。

17.3.2 结合书面指南进行培训

没有人能记住在培训课堂上听过的每一个词。各种测试有助于扩展记忆，但那也是有限的。除了培训，员工需要书面的指导来增强培训效果。事实上，你应该将这些书面指南当作内部培训的一部分。请确保书面指南能够反映人们真正接受的培训。当你更新培训内容时，也要更新这些指南。

> 每次更新培训内容时，请确保每个人都收到更新后的指南副本。此外，不要忽视已工作很久的老员工。任何更新都应该包含对现有员工也进行更新后的培训。有些在公司工作了很久的员工会错过当前的安全策略，并且会因为不了解相关信息而无意导致安全漏洞。

不要专注于让书面指南言辞华丽。你需要的是实用简单的公司策略指导。员工没有时间和耐心去看充满华丽辞藻和行话的指南。简洁地表述内容会更有效。请记住，如今大部分人注意力集中的时间都不超过一分钟，所以如果员工在一分钟之内无法从你创建的指南中发

现答案，他们就不会使用它。

书面指南要尽可能短。用越少的语言传达重要的安全信息，就会有越多的人注意到它。让内容简短并且易于记住。此外，请确保指南有纸质版和电子版。这样做可以让员工将指南放在其他设备（比如智能手机）上，并且可以在任何时候阅读。

17.3.3 创建并使用替代的安全提醒

用户有时候会专注于使用应用程序而不会考虑安全性，除非你提供适当的提醒机制。例如，可以创建带安全提醒的海报并把它们放在公共场所。这个做法看起来像是旧电影里的场景，但它确实是有效的。将安全需求一直摆在用户的面前，即使是在用户休息的时候，是一种保证他们认识到安全重要性的方法。

应用程序也应该包含安全提醒机制。你不能将它们变得烦人。出现“你确定你想要这么做吗”的提醒之后又出现“真的吗”的提醒很令人讨厌，而用户会忽略这些提醒。一种给出安全提醒的有效手段是在每天打开应用程序时出现“今日提示”。不是每一个提示都要聚焦于安全。有时候你可以加入一些有趣的提醒。关键是要让用户去猜猜提醒的内容，这样他们才会继续看这些提醒。

> 曾经有一个富有创造性的安全管理员，他有时候会举行一些竞赛，比如“第一位找出给别人密码时对应的正确安全原则的人将在午餐时获得免费比萨”。这种竞赛不会经常举行，并且几乎不会花费管理层任何开支，但会使员工注意到安全提醒并极大提升公司的安全性。在使人们持续学习安全性知识方面的创造性是使企业保持应用程序及其数据安全的关键。

IT 人员还应该接受人际关系方面的培训。在获得员工关注和合作方面的一个大问题是 IT 人员似乎让大家都觉得自己像傻瓜。如果你经常告诉某些人他们是傻瓜，他们可能会相信你并开始表现得像傻瓜。与不断地跟员工说他们无法理解安全问题相比，正面的强化和用积极的态度帮助他们确实会取得更好的结果。

17.3.4 进行培训有效性检查

你可能会发现尝试让员工关注安全的各种方法都不奏效。当然，问题在于要弄清楚如何改变，从而使安全性得到提升。首先需要知道，安全并非刀枪不入，你能做的就是尽可能地提升安全性并且处理每一个新出现的威胁。但是，你可以尽量接近完美。要达到这一点，一部分工作就是检查你所使用的各种方法的有效性。使用本书介绍过的各种技巧（特别是 15.4 节中提到的技巧）统计什么有效和什么无效。要确定的就是潜在安全问题的原因。

这些安全问题会使你更了解培训是否有效。你也能用本章介绍的技巧随机测试员工。这里的关键不是要找出哪些员工失职，而是要发现当前的培训策略在何处未达到预期效果。当你发现这些问题时，需要考虑如何最好地传达所需的信息，鼓励大家遵守规则，并且确保对故意不遵守流程的行为进行的处罚得以实施。但是，重点要一直放在用更好的培训来确保人们真正明白什么是必需的。

关于作者

John Mueller 是一位自由撰稿人和技术编辑。他的血液里流淌着写作基因，到目前为止，他已经写了 99 本书和 600 多篇文章，主题从网络到人工智能，从数据库管理到埋头编程。他近期出版的图书中包括一本面向 Python 初学者的书、一本面向数据科学家的关于 Python 的书，以及一本关于 MATLAB 的书。他还写了一个 Java 在线学习套件、一本关于使用 JavaScript 进行 HTML5 开发的书，以及一本关于 CSS3 的书。他利用技术编辑技能帮助 60 多位作者完善了手稿内容。John 为 *Data Based Advisor* 和 *Coast Compute* 两本杂志提供了技术编辑服务。请一定要阅读他的博客 http://blog.johnmuellerbooks.com/。

当 John 不在电脑前工作时，你可以在花园里找到他，他会在砍树，或者通常是在享受大自然。他还喜欢酿酒、烘烤饼干和编织。在没有其他事情时，他会做甘油肥皂和蜡烛，这可以在送礼品篮时派上用场。你可以通过电子邮件 john@johnmuellerbooks.com 在网上联系上 John。John 还搭建了一个网站 http://www.johnmuellerbooks.com/。请随意访问并提出改进建议。

关于封面

本书封面上的动物是一只兀鹫（胡兀鹫），也称为髭兀鹰或鱼鹰。兀鹫是一种猛禽，吃腐肉，生活在欧洲南部山区、高加索地区、非洲、印度次大陆和西藏。

作为清道夫，胡兀鹫要靠吃死去动物的遗骸来生存。但是，比起吃肉，这种鸟更喜欢吃骨髓，这占到了它们食物的 85%~90%。偶尔，胡兀鹫会攻击活的动物，并且它们似乎偏爱乌龟。人们已观察到它们会抓乌龟，把乌龟带到高空，然后扔下去，把龟壳摔碎，并吃里面柔软的身体。

胡兀鹫体长 37~49 英寸，体重可以达到 18 磅。雌性通常比雄性略大，而成对的胡兀鹫有助于建造巨大（可以高达 3.3 英尺）的巢。狭长的翅膀使它们很容易区别于其他的秃鹫。胡兀鹫也没有许多其他秃鹫有的秃头，并且这个物种得名于下巴下面长长的黑色羽毛。

胡兀鹫的巨大体型和雄伟的姿态使其在许多社会的神话中占有一席之地。对于伊朗人来说，这种鸟是幸运和快乐的象征，人们相信如果某人在它的阴影下面就预示着这个人可以获得王权。据说，古希腊剧作家埃斯库罗斯的死因是一头“老鹰”将乌龟扔到了他的头上，这头“老鹰”误把他秃顶的头当作一块石头。结合地区、时期和行为，胡兀鹫很可能就是故事中的“老鹰”。在希腊，胡兀鹫是用来预测未来的少数几种鸟类之一，这种行为被称为占卜。最后，在圣经和律法里，胡兀鹫属于禁止人类食用的鸟类。

许多 O’Reilly 图书封面上的动物都是濒危动物，它们对世界很重要。想获取更多关于如何帮助它们的信息，请访问网站 http://animals.oreilly.com。

版权声明

O'REILLY®

Web安全开发指南

Web应用程序的安全性至关重要，病毒、DDoS攻击、中间人攻击、安全漏洞等多种威胁无时无刻不在，任何一种安全事故都可能造成灾难性后果。无论是前端开发人员、网页设计师、用户体验设计师，还是开发运营人员、产品经理、软件工程师，都需具备一定的安全技能，承担起保障Web应用程序及其数据安全的责任。

本书为以上人员提供了具体的Web安全建议以及安全编程示例。书中内容共分为五个部分，分别展示了如何对抗病毒、DDos攻击、安全漏洞和其他恶意入侵，适用于所有平台。

- **为公司制订一份顾及最新设备以及用户需求的安全计划**
- **安全开发技巧实践展示，以及如何有效利用来自库、API和微服务的第三方代码**
- **使用沙盒技术、内部和第三方测试技术，像黑客一样思考**
- **确定何时以及如何更新应用程序软件，制定维护周期**
- **学习如何有效地跟踪安全威胁，确定公司的安全培训需求**

John Paul Mueller，技术编辑、自由作家，写过99本书和600多篇文章，主题涵盖数据库管理、编程、网络技术、人工智能。为*Data Based Advisor*和*Coast Compute*两本杂志提供技术编辑服务，帮助60多位作者完善了手稿。个人网站：www.johnmuellerbooks.com。

“过分思索安全问题会让你完全无法正常工作。在这本书中，John Mueller介绍了一些用于预见和响应Web威胁的指导原则和工具，让你既能避免受骗上当，又不会惊慌失措。”

——Rod Stephens
落基山电脑咨询公司总裁

“John Mueller完整概述了如何快速升级你的安全技能，并介绍了成功的编码实践，这些技术能真正扭转局面，挽救你的公司。”

——Luca Massaron
数据科学家

WEB DEVELOPMENT/SECURITY

封面设计：Randy Comer 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机 / Web开发

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-115-45408-9

ISBN 978-7-115-45408-9
定价：69.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks